UNIT-V
UML
Learning Material

Syllabus:

Advanced Behavioral Modeling: Events and signals, state machines, process and threads, time and space, state chart diagrams.

5.1 Events and Signals

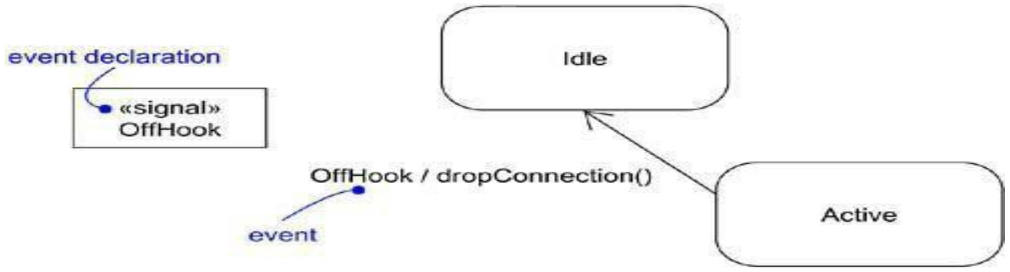
Things that happen are called events, and each one represents the specification of a significant occurrence that has a location in time and space.

In the context of state machines, we use events to model the occurrence of a stimulus that can trigger a state transition.

Events may include signals, calls, the passing of time, or a change in state.

Events may be synchronous or asynchronous, so modeling events is wrapped up in the modeling of processes and threads.

A *signal* is a kind of event that represents the specification of an asynchronous stimulus communicated between instances.



5.1.1 Kinds of Events

Events may be external or internal.

External events are those that pass between the system and its actors.

For example, the pushing of a button and an interrupt from a collision sensor are both examples of external events.

Internal events are those that pass among the objects that live inside the system. An overflow exception is an example of an internal event.

In the UML, we can model four kinds of events: **signals, calls, the passing of time, and a change in state.**

Signals:

A signal represents a named object that is dispatched (thrown) asynchronously by one object and then received (caught) by another.

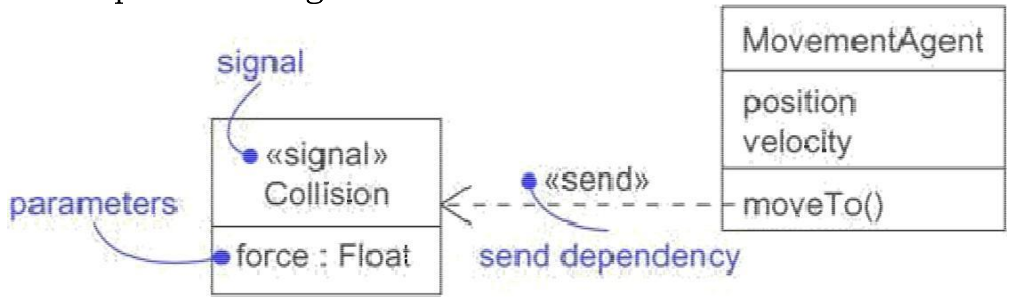
Exceptions are internal signal that we will need to model.

Signals have a lot in common with plain classes. For example, signals may have instances, although we don't generally need to model them explicitly.

Signals may also be involved in generalization relationships, permitting us to model hierarchies of events, some of which are general and some of which are specific also as for classes, signals may have attributes and operations.

A signal may be sent as the action of a state transition in a state machine or the sending of a message in an interaction. The execution of an operation can also send signal.

We can use a dependency, stereotyped as send, to indicate that an operation sends a particular signal.



Call Events:

A call event represents the dispatch of an operation. In both cases, the event may trigger a state transition in a state machine.

Whereas a signal is an asynchronous event, a call event is, in general, synchronous.

when an object invokes an operation on another object that has a state machine, control passes from the sender to the receiver, the transition is triggered by the event, the operation is completed, the receiver transitions to a new state, and control returns to the sender.

Modeling a call event is indistinguishable from modeling a signal event. In both cases, we show the event, along with its parameters, as the trigger for a state transition.

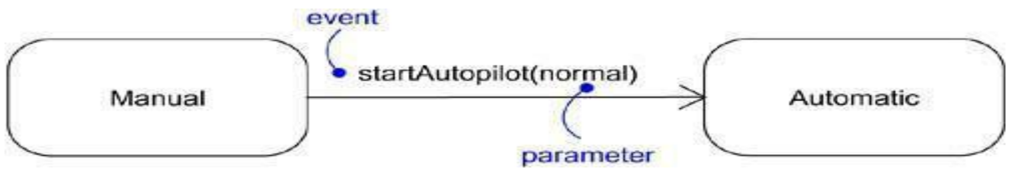


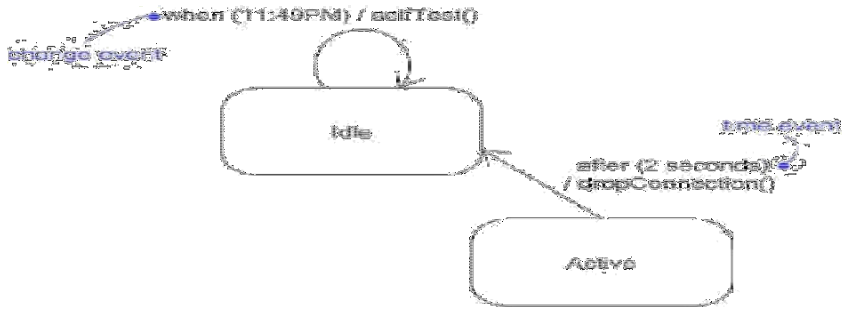
Fig: **Call Events**

Time and Change Events:

A time event is an event that represents the passage of time.

In the UML we model a time event by using the keyword **after** followed by some expression that evaluates to a period of time. Such expressions can be simple (after 2 seconds)or complex. Unless we specify it explicitly, the starting time of such an expression is the time since entering the current state.

A change event is an event that represents a change in state or the satisfaction of some condition. In the UML we model a change event by using the keyword **when** followed by some Boolean expression. We can use such expressions to mark an absolute time (such as when time = 11:59) or for the continuous test of an expression (for example, when altitude < 1000).



Sending and Receiving Events:

Signal events and call events involve at least two objects:

1. The object that sends the signal or invokes the operation

2. The object to which the event is directed.

   Any instance of any class can send a signal to or invoke an operation of a receiving object.

   When an object sends a signal, the sender dispatches the signal and then continues along its flow of control, not waiting for any return from the receiver. In contrast, when an object calls an operation, the sender dispatches the operation and then waits for the receiver.

Any instance of any class can receive a call event or a signal.

   If this is a synchronous call event, then the sender and the receiver are in a rendezvous for the duration of the operation. This means that the flow of control of the sender is put in lock step with the flow of control of the receiver until the activity of the operation is carried out.

   If this is a signal, then the sender and receiver do not rendezvous: the sender dispatches the signal but does not wait for a response from the receiver

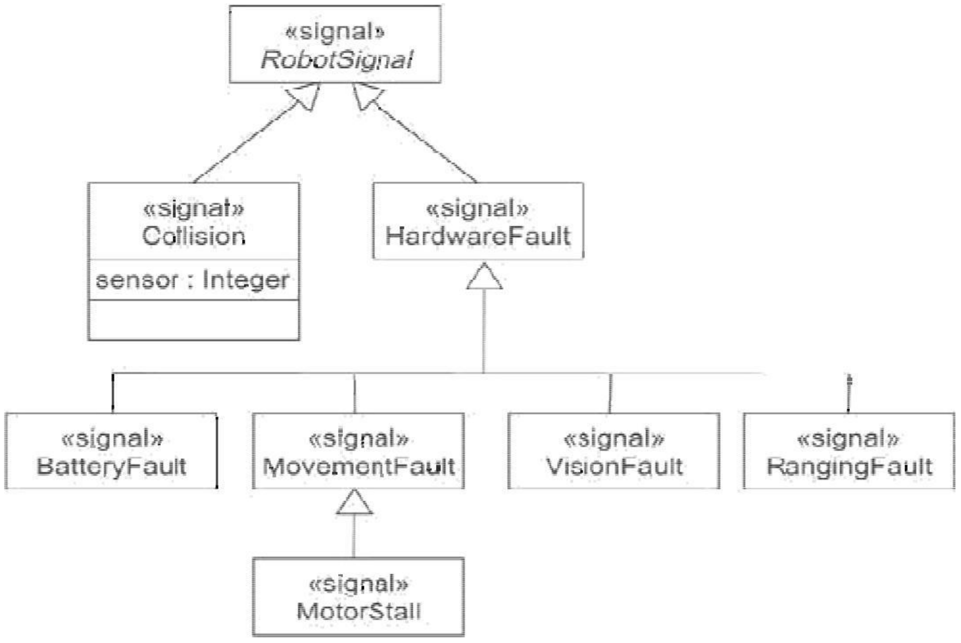
Common Modeling Techniques

1. Modeling a Family of Signals:

   Consider all the different kinds of signals to which a given set of active objects may respond.

   Look for the common kinds of signals and place them in a generalization/specialization hierarchy using inheritance. Elevate more general ones and lower more specialized ones.

   Look for the opportunity for polymorphism in the state machines of these active objects. Where we find polymorphism, adjust the hierarchy as necessary by introducing intermediate abstract signals.

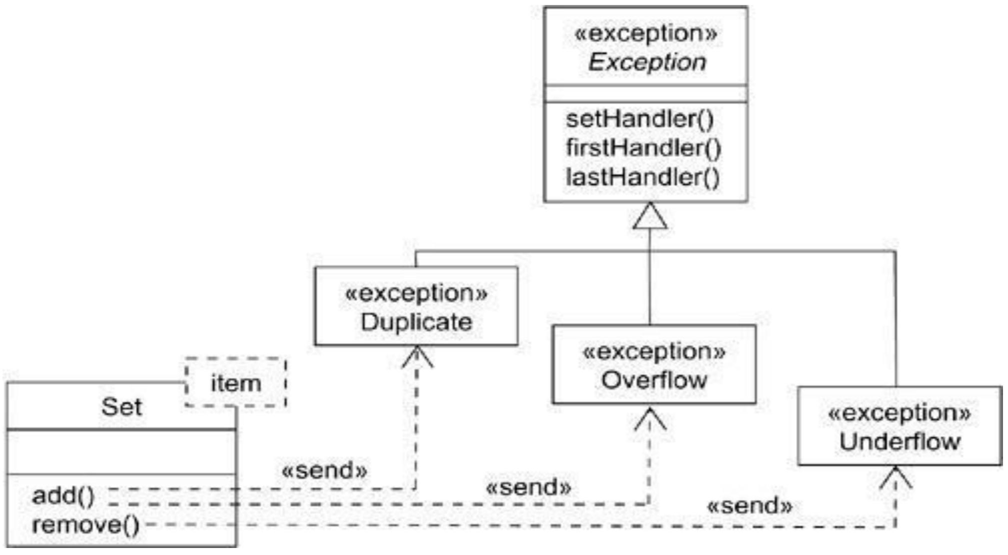


2. Modeling Exceptions:

For each class and interface, and for each operation of such elements, consider the exceptional conditions that may be raised.

Arrange these exceptions in a hierarchy. Elevate general ones, lower specialized ones, and introduce intermediate exceptions, as necessary.

For each operation, specify the exceptions that it may raise. We can do so explicitly (by showing send dependencies from an operation to its exceptions) or we can put this in the operation's specification



5.2 State Machines

We use an **interaction** to model the behavior of a society of objects that work together.

We use a **state machine** to model the behavior of an individual object.

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

A *state* is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.

An *event* is the specification of a significant occurrence that has a location in time and space. In the context of state machines, an event is an occurrence of a stimulus that can trigger a state transition.

A *transition* is a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and specified conditions are satisfied.

An *activity* is ongoing non atomic execution within a state machine.

An *action* is an executable atomic computation that results in a change in state of the model or the return of a value.

Graphically, a state is rendered as a rectangle with rounded corners. A transition is rendered as a solid directed line

We use state machines to model the dynamic aspects of a

system. We can visualize a state machine in two ways:

1. By emphasizing the flow of control from activity to activity (using activity diagrams).
2. By emphasizing the potential states of the objects and the transitions among those states (using state chart diagrams).

The UML provides a graphical representation of states, transitions, events, and actions. This notation permits we to visualize the behavior of an object in a way that emphasizes the important elements in the life of that object.

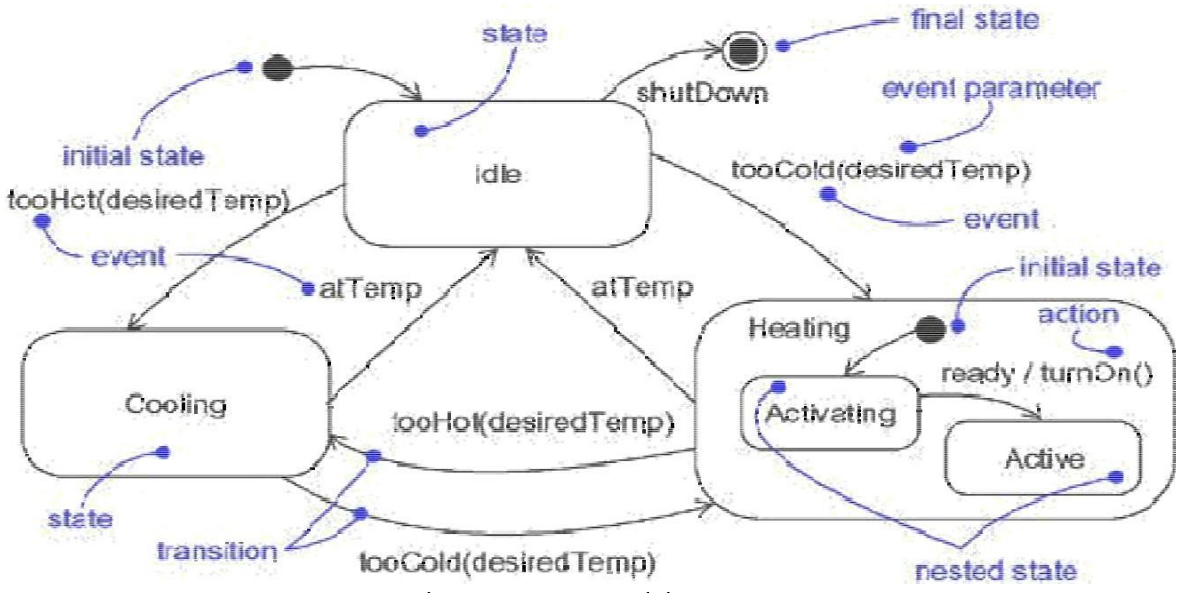


Fig: State Machines

5.2.1 States:

A state is a condition or situation during the life of an object during which it satisfies some condition, performs some activity, or waits for some event.

An object remains in a state for a finite amount of time.

Example: a Heater in a home might be in any of four states:

Idle (waiting for a command to start heating the house)
Activating (its gas is on, but it's waiting to come up to temperature) Active (its gas and blower are both on)
ShuttingDown (its gas is off but its blower is on, flushing residual heat from the system).

A state has several parts:

1. Name: A textual string that distinguishes the state from other states; a state may be anonymous, meaning that it has no name
2. Entry/exit actions: Actions executed on entering and exiting the state, respectively
3. Internal Transitions: Transitions that are handled without causing a change in state
4. Substates: The nested structure of a state, involving disjoint (sequentially active) or concurrent (concurrently active) substates
5. Deferred Events: A list of events that are not handled in that state but, rather, are postponed and queued for handling by the object in another state

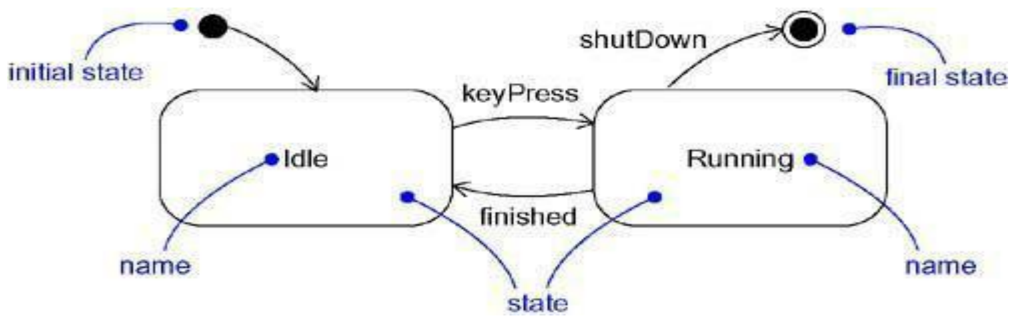


Fig: States

Initial and Final States:

There are two special states that may be defined for an object's state machine.

First, there's the initial state, which indicates the default starting place for the state machine or substate. An initial state is represented as a filled black circle.

Second, there's the final state, which indicates that the execution of the state machine or the enclosing state has been completed. A final state is represented as a filled black circle surrounded by an unfilled circle.

5.2.2 Transitions:

A transition is a relationship between two states indicating that an object in the first state will perform certain actions and enter in to the second state when a specified event occurs and specified conditions are satisfied.

On such a change of state, the transition is said to fire. Until the transition fires, the object is said to be in the source state; after it fires, it is said to be in the target state.

Example: a Heater might transition from the Idle to the Activating state when an event such as tooCold (with the parameter desiredTemp) occurs.

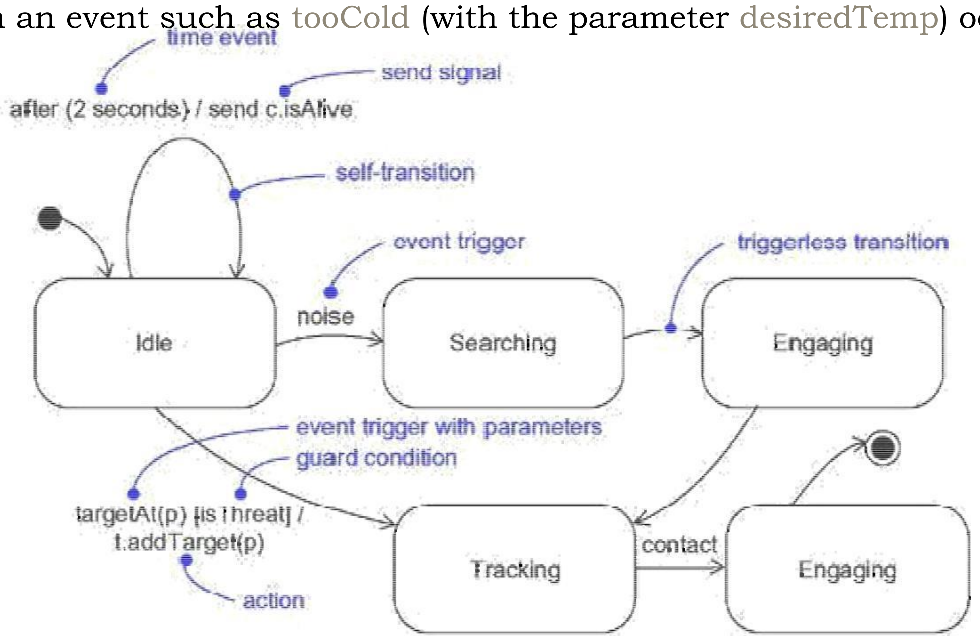


Fig: Transitions

<u>A transition has five parts:</u>

1. **Source state**: The state affected by the transition; if an object is in the source state, an outgoing transition may fire when the object receives the trigger event of the transition and if the guard condition, if any, is satisfied
2. **Event trigger**: The event whose reception by the object in the source state makes the transition eligible to fire, providing its guard condition is satisfied.
3. **Guard condition**: A Boolean expression that is evaluated when the transition is triggered by the reception of the event trigger; if the expression evaluates True, the transition is Guard eligible to fire; if the expression evaluates False, the transition does not fire and if condition there is no other transition that could be triggered by that same event, the event is lost.
4. **Action**: An executable atomic computation that may directly act on the object that owns the state machine, and indirectly on other objects that is visible to the object.

5. **Target state**: The state that is active after the completion of the transition.

   A transition is rendered as a solid directed line from the source to the target state. A self-transition is a transition whose source and target states are the same.

A transition may have multiple sources as well as multiple targets.

5.2.3 Advanced States and Transitions:

The UML's state machines have a number of advanced features that help us to manage complex behavioral models.

These features often reduce the number of states and transitions and codify a number of common and somewhat complex idioms.

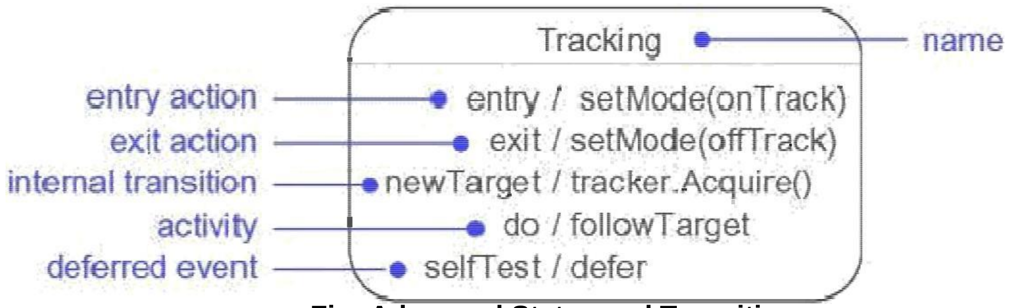


Fig: Advanced States and Transitions

Entry and Exit Actions

In a number of modeling situations, we want to dispatch the same action whenever we enter a state, no matter which transition led we there. Similarly, when we leave a state, we want to dispatch the same action no matter which transition led us away.

Entry and exit actions may not have arguments or guard conditions.

Internal Transitions

Once inside a state, we will encounter events when we want to handle without leaving the state. These are called internal transitions, and they are subtly different from self-transitions.

Internal transitions may have events with parameters and guard conditions.

Activities

When an object is in a state, it generally sits idle, waiting for an event to occur. Sometimes, however, we may wish to model an ongoing activity. While in a state, the object does some work that will continue until it is interrupted by an event.

Use the special do transition to specify the work that's to be done inside a state after the entry action is dispatched. The activity of a do transition might name another state machine (such as followTarget).

Deferred Events

In every modeling situation, we want to recognize some events and ignore others. In some modeling situations, we want to recognize some events but postpone a response to them until later.

A deferred event is a list of events whose occurrence in the state is postponed until a state in which the listed events are not deferred becomes active, at which time they occur and may trigger transitions as if they had just occurred. As we can see in the previous figure, we can specify a deferred event by listing the event with the special action defer.

5.2.4 Substates :

A substate is a state that's nested inside another one.

The UML's state machines – substates, that does even more to help we simplify the modeling of complex behaviors.

A simple state is a state that has no substructure. A state that has substate that is, nested states is called a composite state.

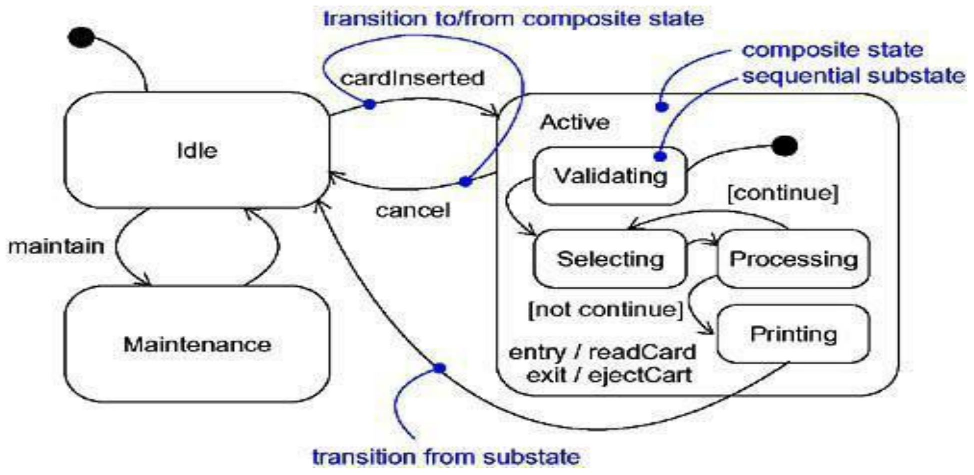
A composite state may contain either concurrent (orthogonal) or sequential (disjoint) substates.

Sequential Substates:

Sequential substates partition the state space of the composite state into disjoint states.

A nested sequential state machine may have at most one initial state and one final state.

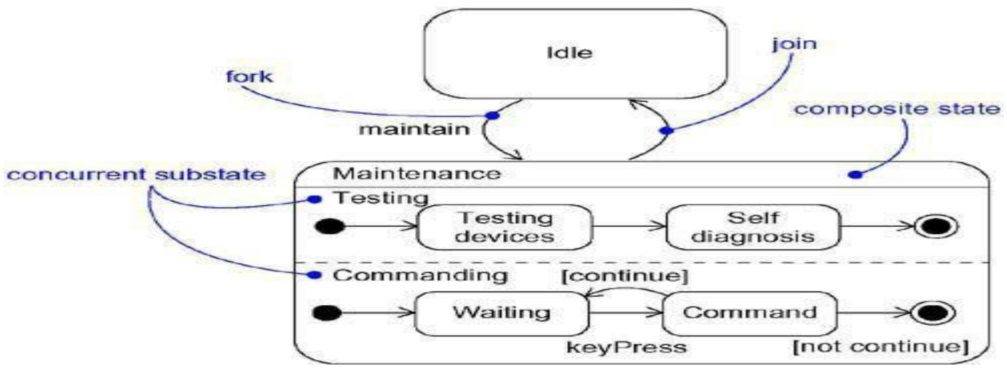
For example Substates such as Validating and Processing are called sequential, or disjoint, substates



Concurrent Substates:

Concurrent substates to specify two or more state machines that execute in parallel in the context of the enclosing object

A nested concurrent state machine does not have an initial, final, or history state. However, the sequential substates that compose a concurrent state may have these features.



Common Modeling Techniques:

1. **Modeling the Lifetime of an Object:** model the lifetime of an object, we essentially specify three things: the events to which the object can respond, the response to those events, and the impact of the past on current behavior. Modeling the lifetime of an object also involves deciding on the order in which the object can meaningfully respond to events, starting at the time of the object's creation and continuing until its destruction.

Set the context for the state machine, whether it is a class, a use case, or the system as a whole.

1. If the context is a class or a use case, collect the neighboring classes, including any parents of the class and any classes reachable by associations or dependences. These neighbors are candidate targets for actions and are candidates for including in guard conditions.

2. If the context is the system as a whole, narrow your focus to one behavior of the system. Theoretically, every object in the system may be a participant in a model of the system's lifetime, and except for the most trivial systems, a complete model would be intractable.

Establish the initial and final states for the object. To guide the rest of your model, possibly state the pre- and postconditions of the initial and final states, respectively.

Decide on the events to which this object may respond. If already specified, you'll find these in the object's interfaces; if not already specified, you'll have to consider which objects may interact with the object in your context, and then which events they may possibly dispatch.

Starting from the initial state to the final state, lay out the top-level states the object may be in. Connect these states with transitions triggered by the appropriate events. Continue by adding actions to these transitions.

Identify any entry or exit actions (especially if you find that the idiom they cover is used in the state machine).

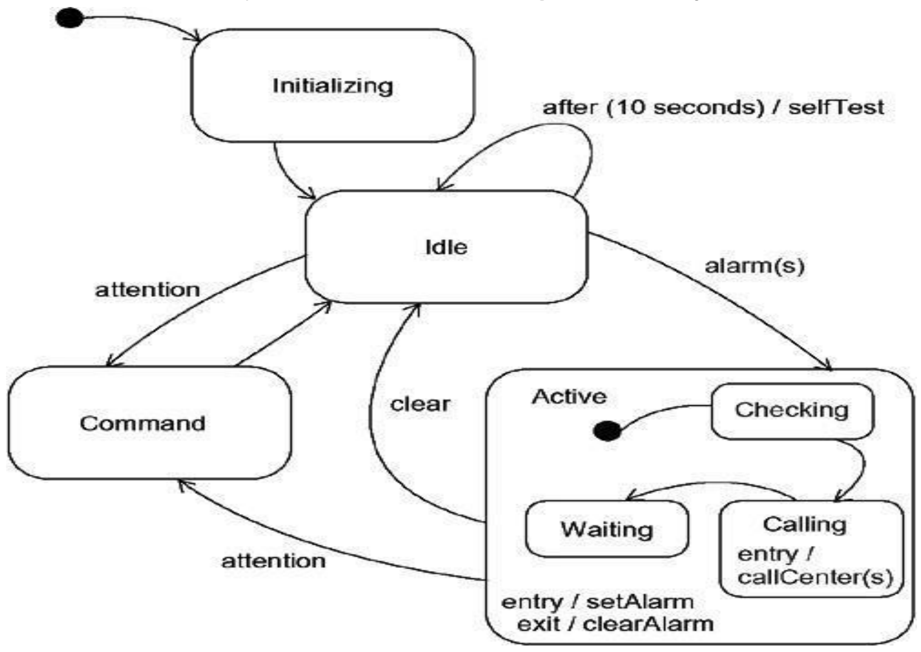
Expand these states as necessary by using substates.

Check that all events mentioned in the state machine match events expected by the interface of the object. Similarly, check that all events expected by the interface of the object are handled by the state machine. Finally, look to places where you explicitly want to ignore events.

Check that all actions mentioned in the state machine are sustained by the relationships, methods, and operations of the enclosing object.

Trace through the state machine, either manually or by using tools, to check it against expected sequences of events and their responses. Be especially diligent in looking for unreachable states and states in which the machine may get stuck.

After rearranging your state machine, check it against expected sequences again to ensure that you have not changed the object's semantics.



5.3 Process and threads

In uml , each independent flow of control is modeled as a active object.

- An active object is a process or thread that can initiate control activity.
- An active object is an instance of an active class.
- Active objects can communicate with one another by passing messages.
- Message passing must be extended with certain concurrency.
- An active class is a class whose instances are active objects.

Definition:

Process:

A process is a heavyweight flow that can execute concurrently with other processes.

Thread:

An active object is an object that owns a process or thread and can initiate control activity.

- Graphically, an active class is rendered as a rectangle with thick lines. Processes and threads are rendered as stereotyped active classes.

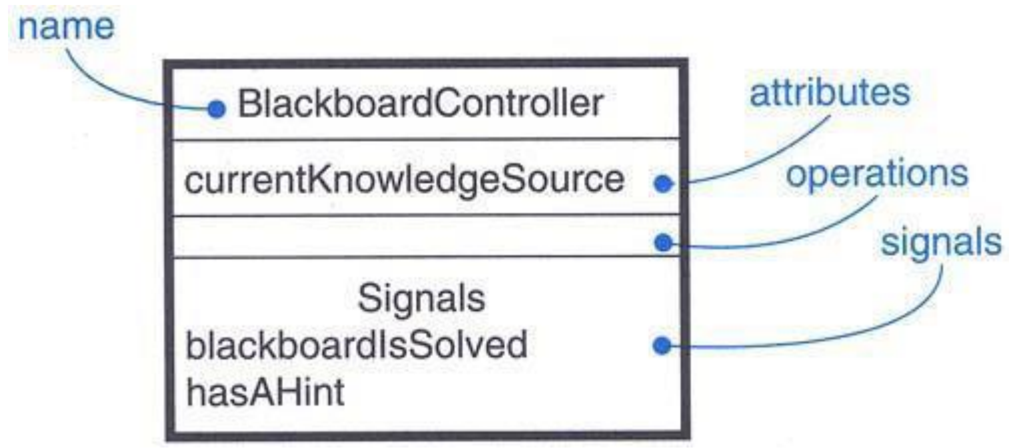


Fig: active class

Flow of control:

In a sequential system, there is a single flow of control. i.e, one thing, and one thing only, can take place at a time.

In a concurrent system, there is multiple simultaneous flow of control i.e, more than one thing can take place at a time.

Classes and Events

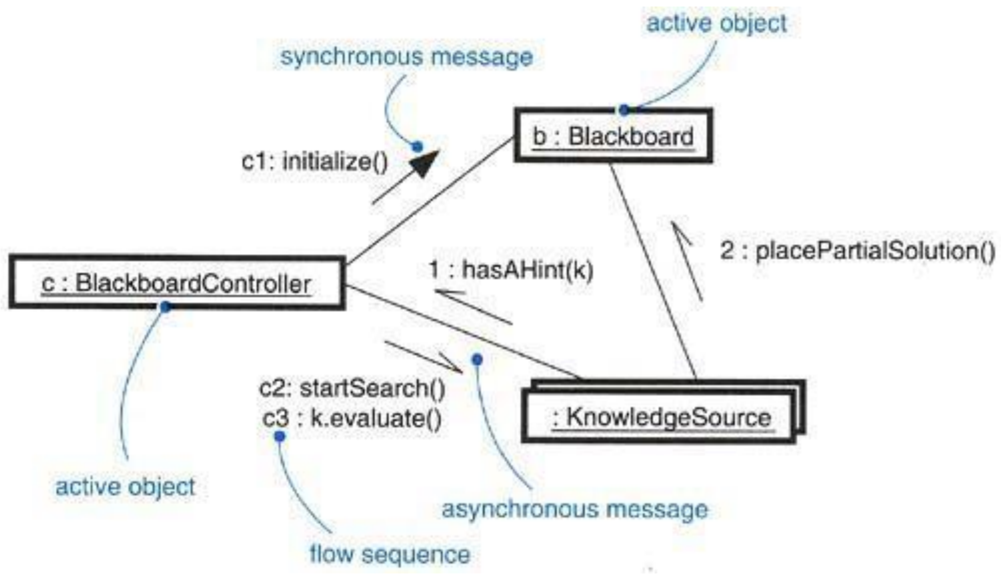
- Active classes object is created, the associated flow of control is started; when the active object is
- with other processes destroyed are just classes which represents an independent flow of control
- Active classes share the same properties as all other classes.
- When an active, the associated flow of control is terminated
- *two standard stereotypes* that apply to active classes are, ***<<process>>*** – Specifies a heavyweight flow that can execute concurrently. (heavyweight means, a thing known to the OS itself and runs in an independent address space) ***<<thread>>*** – Specifies a lightweight flow that can execute concurrently with other threads within the same process (lightweight means, known to the OS itself.)

Communication:

- In a system with both active and passive objects, there are *four possible combinations of interaction.*
  First: A message may be passed from one passive object to another.
- **Second**: A message may be passed from one active object to another.
- **Third:** A message may be passed from one active object to a passive object.

- **Fourth: A** message may be passed from one passive object to an active object.
- **In** *inter-process communication* there are two possible styles of communication.
- ***First:***: one active object might synchronously call an operation of another.
- **Second**: one active object might asynchronously send a signal or call an operation of another object.
- A synchronous message is rendered as a full arrow and an asynchronous message is rendered as a half arrow.

Fig: communication

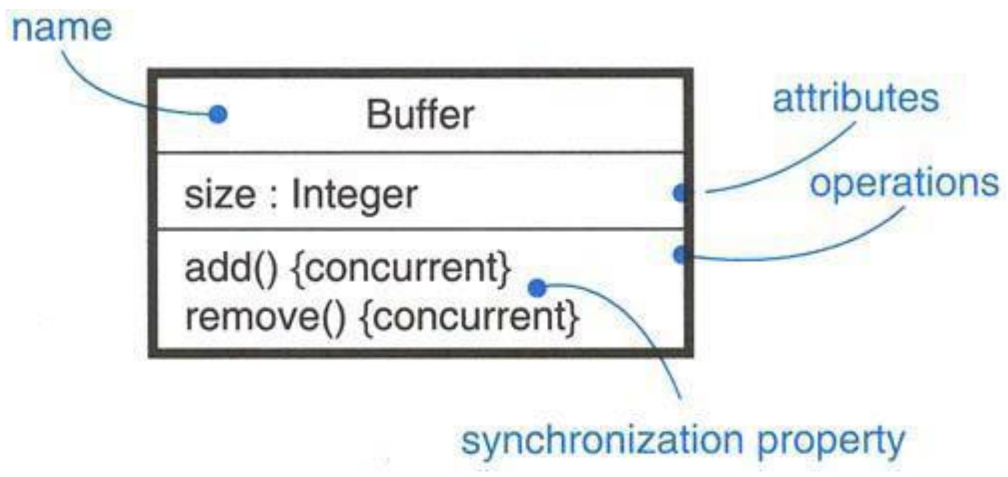


Synchronization:

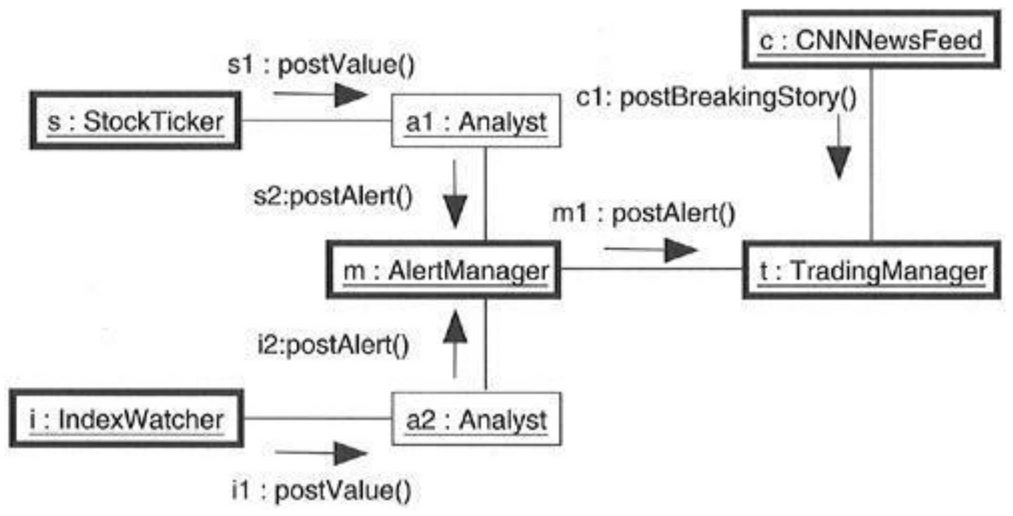
- synchronization means arranging the flow of controls of objects so that mutual exclusion will be guaranteed.
- in object-oriented systems these objects are treated as a critical region

Three synchronization approaches:

- *Sequential* – Callers must coordinate outside the object so that only one flow is in the object at a time
- *Guarded* – multiple flow of control is sequentialized with the help of object's guarded operations. in effect it becomes sequential.
- *Concurrent* – multiple flow of control is guaranteed by treating each operation as atomic.
- synchronization are rendered in the operations of active classes with the help of constraints.



Modeling multiple flow of control:
Figure shows part of the process view of a trading system.



Modeling inter-process communication:

- Consider which of these active objects represent processes and which represent threads. Distinguish them using the appropriate stereotype.
- Informally specify the underlying mechanism for communication by using notes, or more formally by using collaborations.

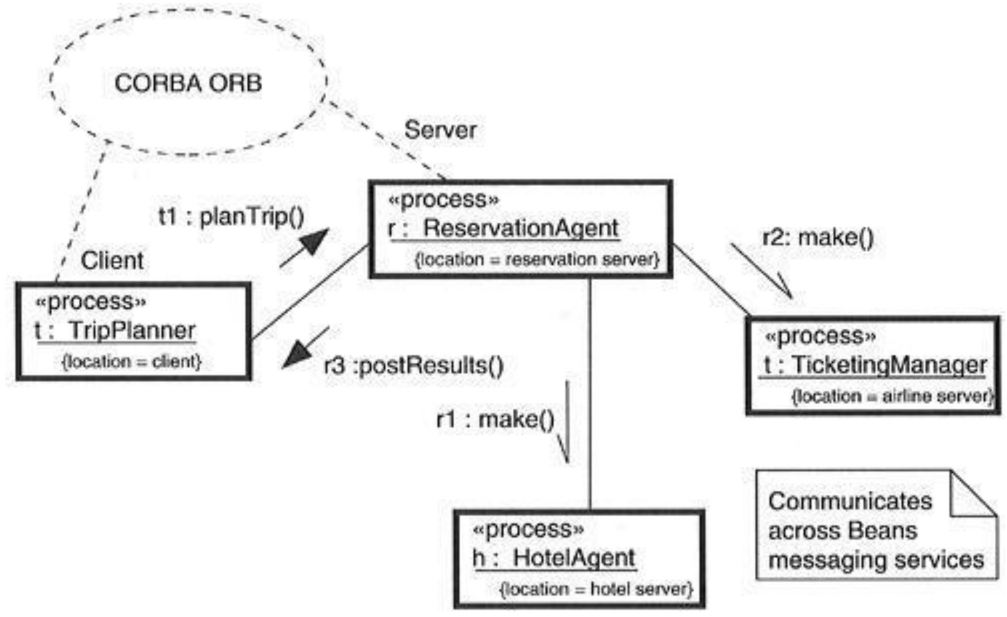


Figure shows a distributed reservation system with processes spread across four nodes.

Model messaging using asynchronous communication model remote produce calls using synchronous communication.

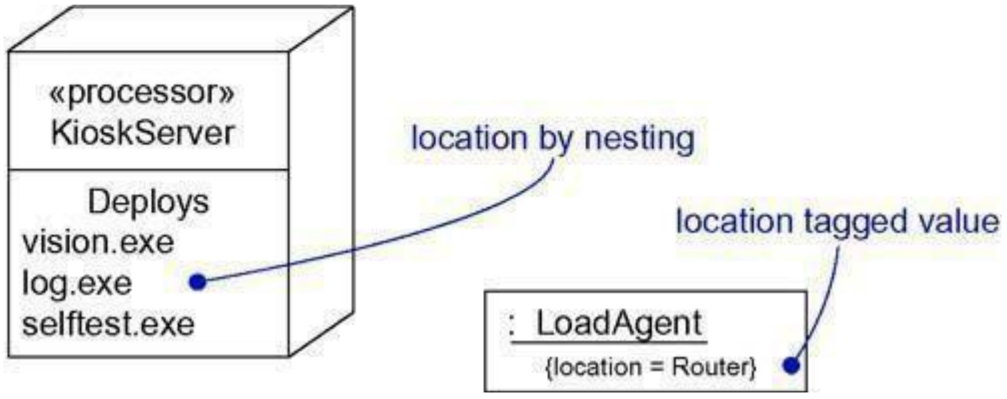
5.4 TIME AND SPACE :

Modeling time and space is an essential element of any real time and distributed system. A real time system is any information processing system that operates with a time constraint.

- A distributed system is a model in which components located on net worked computers communicate and co-ordinate their actions by passing messages .The components interact with each other in order to achieve a common goal. The uml provides a graphic representation for timing marks,time expressions ,timing constraints and location.
- **To** represent the modeling needs of real time and distributed systems, the uml provides a graphical representation for timing marks , time expression ,time constraints and location.
  Time :
  A timing mark denotes the time at which an event occurs. A timing mark is formed as an expression from the name given to the message.
- A time expression is an expression that evaluates to an absolute or relative value of time
- A timing constraint is an sematic statement about the relative or absolute value of time.

- Graphically, at timing constraint is rendered as for any constraint i.e. A strong enclosed by brackets and generally connected to an element by a dependency relationship.
- Location is the placement of a component on a node Graphically location is rendered as a tagged value i.e a string enclosed by brackets and placed below on elements name, or as the nesting of components inside nodes.
  For example: figure shows the executable component vision.exe my reside on the node named kiosk server an instance of the class load agent lives on the node named router.
  you can model the location of an element in two ways in the uml
  'First: as shown for the kiosk server ,you can physically nest the element (textually or graphically) in a extra compartment in its enclosing node

fig:location



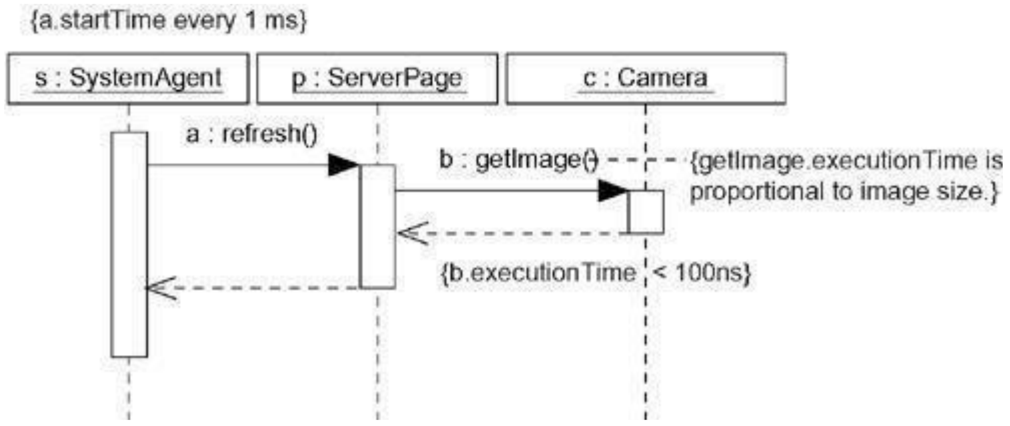
In the uml you model the deployment view of a system by using deployment diagrams that represent the processors and devices on which your system executes.

common modeling techniques:

To model timing constraints:

For example, as shown in Figure the left-most constraint specifies the repeating start time the call event **refresh.**, the center timing constraint specifies the maximum duration for calls to **getImage.** Finally, the right-most constraint specifies the time complexity of the call event **getImage.**

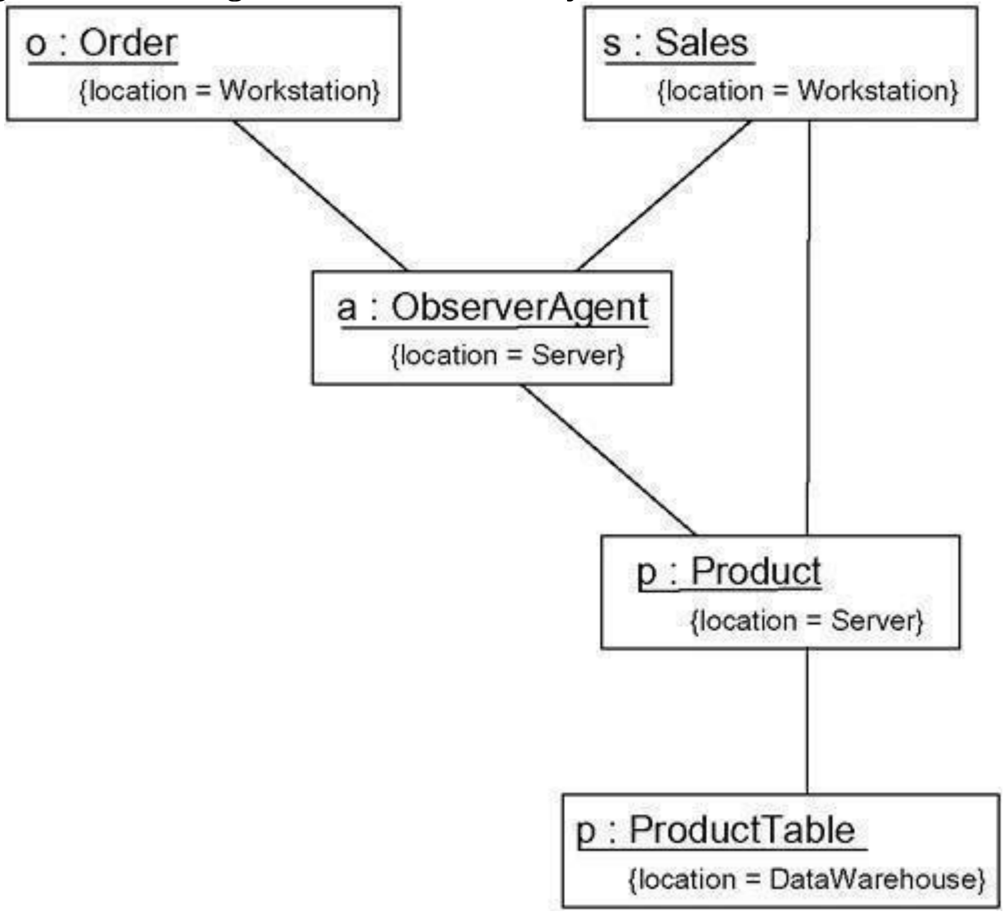
Figure Modeling Timing Constraint



To model the distribution of objects:

Figure provides an object diagram that models the distribution of certain objects in a retailsystem. The value of this diagram is that it lets you visualize the physical distribution of certain key objects. As the diagram shows, two objects reside on a **Workstation** (the **Order** and **Sales** objects), two objects reside on a **Server** (the**ObserverAgent**and the **Product** objects), and one object resides on a **DataWarehouse** (the **ProductTable** object).

figure to modeling the distribution of objects:



state chart diagrams:

we will use state chart diagrams to model the dynamic aspects of a system.

A state chart diagram shows a state machine. An activity diagram is a special case of state chart diagram in which all or most of the states are activity states and all or most of transitions are trigged by completion of activities in the source state.

- Note: both activity and state chart diagrams are useful in modelling the life time of an object. However , an activity diagram shows flow of control from activity to activity where as state chart diagram shows flow of control from state to state.

Terms and concepts:

definition:

A *state chart diagram* shows a state machine, emphasizing the flow of control from state to state.

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.

A *state* is a condition or situation in the life of an object during which it satisfies some condition, performs some activity, or waits for some event. An *event* is the specification of a significant occurrence that has a location in time and space.
In the context of state machines, an event is an occurrence of a stimulus that can trigger a state transition.

A *transition* is a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state when a specified event occurs and specified conditions are satisfied.

An *activity* is ongoing non atomic execution within a state machine. An *action* is an executable atomic computation that results in a change in state of the model or the return of a value.

Graphically, a state chart diagram is a collection of vertices and arcs.

contents:

state chart diagram commonly contains

simple state
composite state
transitions including events and actions.

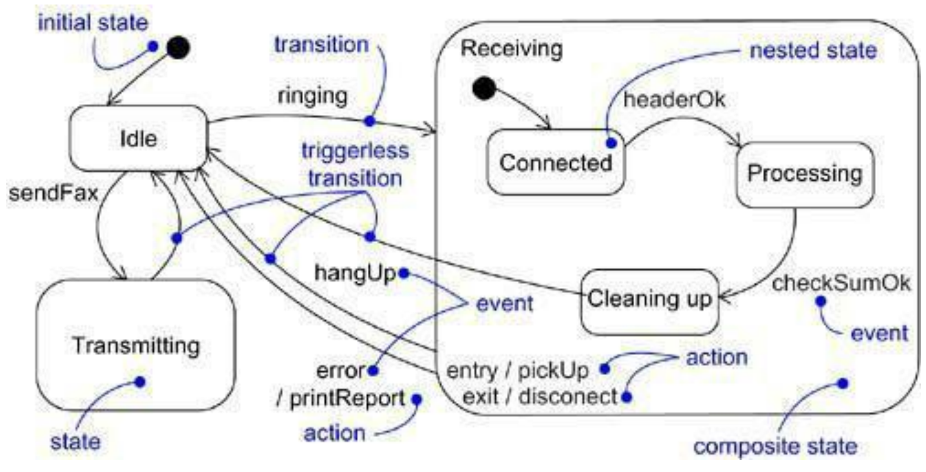
5.5 State Chart Diagrams:

Statechart diagrams are one of the five diagrams in the UML for modeling the dynamic aspects of systems. A statechart diagram shows a state machine.

A *statechart diagram* shows a state machine, emphasizing the flow of control from state to state.

A *state machine* is a behavior that specifies the sequences of states an object goes through during its lifetime in response to events, together with its responses to those events. A *state* is a condition or situation in the life of an object during which it satisfies some condition, performs some activity, or waits for some event.

A *transition* is a relationship between two states indicating that an object in the first state will perform certain actions and enter the second state.



Statechart diagrams commonly contain

Simple states and composite states

Transitions, including events and actions

Statechart diagrams may contain branches, forks, joins, action states, activity states, objects, initial states, final states, history states, and so on. Indeed, a statechart diagram may contain any and all features of a state machine.

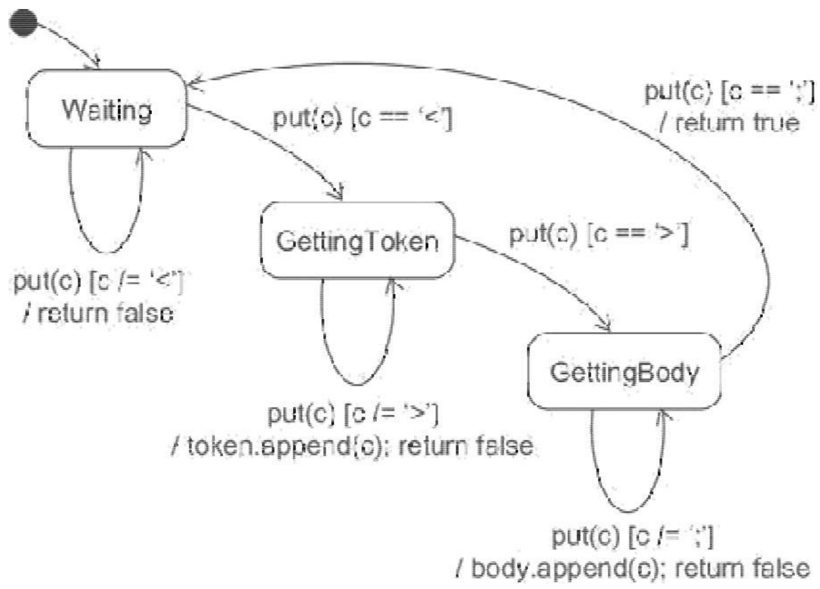
Common Modeling Techniques:

Modeling Reactive Objects:

Model the behavior of a reactive object, essentially specify three things:

- The stable states in which that object may live,
- The events that trigger a transition from state to state,
- The actions that occur on each state change.

Modeling the behavior of a reactive object also involves modeling the lifetime of an object, starting at the time of the object's creation and continuing until its destruction, highlighting the stable states in which the object may be found.



To model a reactive object,

Choose the context for the state machine, whether it is a class, a use case, or the system as a whole.

Choose the initial and final states for the object. To guide the rest of our model, possibly state the pre-condition and post-conditions of the initial and final states, respectively.

Decide on the stable states of the object by considering the conditions in which the object may exist for some identifiable period of time. Start with the high-level states of the object and only then consider its possible substates.

Decide on the meaningful partial ordering of stable states over the lifetime of the object.

Decide on the events that may trigger a transition from state to state. Model these events as triggers to transitions that move from one legal ordering of states to another.

Attach actions to these transitions (as in a Mealy machine) and/or to these states (as in a Moore machine).

Consider ways to simplify our machine by using substates, branches, forks, joins, and history states.

Check that all states are reachable under some combination of events.

Check that no state is a dead end from which no combination of events will transition the object out of that state.

Trace through the state machine, either manually or by using tools, to check it against expected sequences of events and their responses.

1. A) What is event and signal? Explain with an example?

   B) How to model a family of signals?

2. What is a domain state model? Explain nested state with appropriate diagram?
3. Explain sub states? Explain about Sequential sub states, concurrent sub states.
4. Explain about process and threads with examples.
5. Explain about Time and space with examples.
6. Describe in detail about state chart diagrams with an example.
7. Define state, events and transitions. Draw state machine diagram for ATM application.

II.B.Tech-I-Semester A.Y.2017-18 IT

II) Descriptive Questions