UNIT 4 ADVANCED BEHAVIORAL MODELING USING UML

Structure **Page no.**

4.1 INTRODUCTION

Today, most organizations depend on software to perform dynamic activities. Insurance, E-Procurement, E-Commerce, E-Education, Infrastructure development, government offices etc. take a global approach and incorporate the software in the business to increase the profit. Due to dynamic requirements in business processes, the requirement deals with the changes in the system behavior. Various behavior models are explained with the help of interactive diagrams and activity diagrams. These behavior models perform the integration of software with the system behavior. The behavior models of the system narrate the internal parameter of the entire system as explained in the previous chapter behavior modeling.

The behavioral model is derived from the current fields, which can predict various actions. UML behavioral diagrams demonstrate the components of a system that are time-dependent and communicate the dynamic behavior and their interactions.

This unit provides various existing event approaches to behavioral modeling. UML is used to define a wide variety of behavior using the basic features of events, states, signals, and transactions. Using these features state machines, and the time-space diagram represents arcs (transactions) and vertices (states) in the behavioral models. The relationship with structural elements in these diagrams will be explained in this unit.

Various behavioral diagrams are described by the UML, which are as follows:

- Events and Signals Diagram
- State Machine Diagram
- Processes and Threads
- Timing Diagram
- Sequence Diagram
- Communication Diagram

4.2 OBJECTIVES

After going through this unit you will be able to:

- Explain the structure and relationship between objects with the help of events,
- Describe the behavior of objects and the state machine,
- To monitor the operation of the process with process and threads,
- Explain the resource and time, and
- To understand the different states of objects and the state chart diagram.

4.3 EVENTS AND SIGNALS

In real-world situations, mostly things are happened at the same time or in an unpredictable manner. A system can understand its static structure using objects. The object's structure and its relationship define the class model of a particular moment in time. The changes of the model and their relationship depend on the state of the model. Suppose Train No. 04010 should depart from New Delhi station before it can arrive in Lucknow station; the two events are causally related. Similarly, Train No. 04010 may depart before or after Train No. 04050 departs from Mumbai station; the two events are causally unrelated.

4.2.1 Events

Events are also known as the "Things that occurred". UML, event things happened in the form of the model. An event is a model specification that causes something to happen in time and space.

- Events can be internal or external.
- Events may occur in synchronous and asynchronous forms.
- The time at which the event executes is an implicit attribute of the event.
- One event may occur before or follow another, or the two events may be unrelated.
- Two events that are causally unrelated is known as concurrent event.
- If the communication delay between two locations is greater than the difference in event times, then the events must be concurrent.

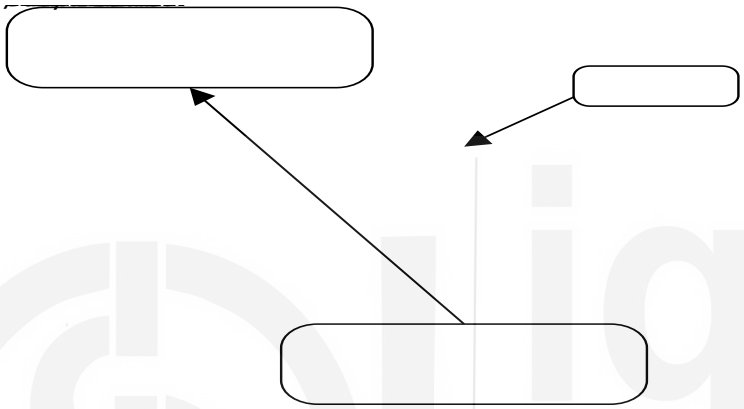

Figure 4.1: Events

There are various types of events:

1. Signal Event:

It is a way of expression, transmission of information from one object to another object.

- A signal event happens when two or more items happen at the same time.
- Various characteristics, instances, relationships, and operations are associated with a signal event.
- A signal's attributes serve as a parameter.
- A signal event is used for sending and receiving a signal.
- It is a message between objects, while a signal event is an occurrence in time.
- The unique signal class defines every signal transmission, and each signal class has a unique name to indicate the common structure and behaviour.

2. Call Event

It refers to an operation transmitted from one object to another. It is used as a trigger to change the state of a state machine. In this event, the sender sends a message and waits for the acknowledgement from the receiver's end. For example, a customer or user wait for the OTP at the time of some online transactions. In figure 4.2, events are defined by remote control used as manual or automatic.

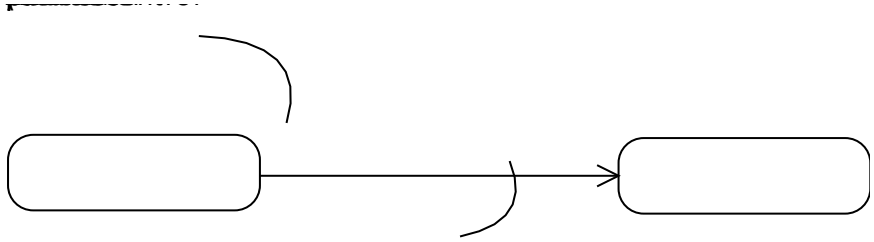

Figure 4.2: Call Events

3. Time Event and Change Event

It is an event that occurs at an accurate time or the passage of a given time interval. It is denoted by the "after" followed by the parenthesis expression that measures the time duration. In figure 4.3, when the time event is executed after (15 seconds) call will be disconnected. A change event is used to denote an event that denotes a change of state or fulfills some of the boolean conditions. The change event is denoted by "when" followed by a boolean expression in parenthesis. When time =11:50 AM, the event executes and stops the charging.

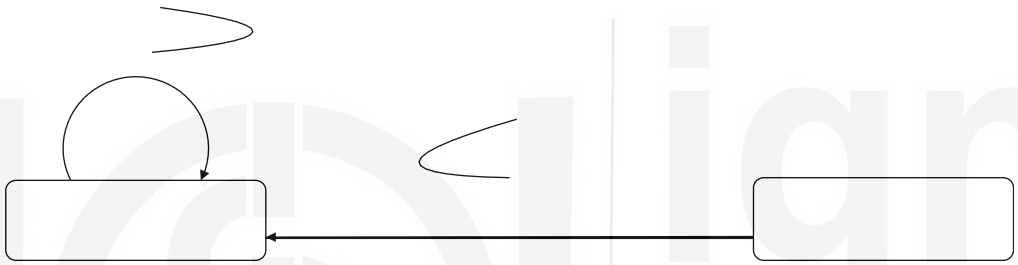

Figure 4.3: Time and Change Event

4. Sending and Receiving Event

Every instance of a class has either a call event or a signal event. The sender is waiting for the receiver if the call is synchronous. If the signal is used, it works at the end of the receiver without anything.

At least two objects are involved in the calling and signaling event: the object on which the event has been directed for transmitting or initiating the operation. Any class instances (objects) are capable of sending or invoking a signal for a receiving object. When an entity passes a signal, the user transmits the message and afterward proceeds with its control flow without waiting for the receiver to respond. For example, when you send a signal push button to the actor communicating with an ATM device. Users can proceed on their way regardless of whether the system is sending the signal to the user or not. In comparison, the sender dispatches the operation and waits for the recipient in the process when the object calls an operation.

A call event or a signal can be sent to any instance in any class. If the sender and the recipient have made an appointment for the period of the transaction, then it indicates the sender's control flow is locked up with the receiver's control flow before the process is completed. If that is a signal, the receiver and the sender will not meet: the sender sends out the signal but will not wait for the recipient to respond. In any situation, this event will be missed or lost (When no response to the event is provided), the receiver state machine may be activated, or a standard method call may be invoked in this situation.

In the UML, events of an object receive are modeled as operations in the object class, and the named signals that an object may receive are modeled in the UML by naming them in an extra row of the class, as shown in figure 4.4.

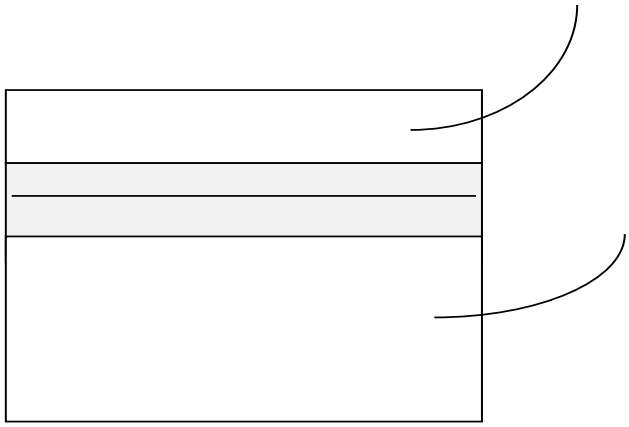

Figure 4.4: Signals and Active Classes

4.2.2 Signals

A signal is used to present an object which is transmitted by one object and received by another object asynchronously. Every signal has instances, attributes, and relationships that perform various operations. A class instance can receive a class or signal event. If the event is a synchronous call event, the sender and receiver are met during the operation. It means the flow of control of the sender is suspended from the receiver flow of the control until the operation is terminated. In this case, the sender and receiver do not lock. The sender sends the signal continuously without the wait a response from the receiver end. In such cases, the event may be lost, it may activate the receiver's state machine, or it may invoke a call method. For example, the event is an email sent by one user and received by another user.

4.2.3 Common Modeling Techniques

- **Modeling a Signal Family**

To build a signal family model:

- Consider all types of signals that can be received by a given collection of active objects.
- Search for the common types of signals and put them in an inheritance generalization/specialization hierarchy.
- Search for polymorphism in these active objects' state machines. Adjust the hierarchy when you find polymorphism by introducing intermediary abstract signals.

Signal events are hierarchical in most event-driven systems. For example, mobile signals can differentiate between external signals like a network and internal signals like a hardware fault. However, external and internal signals do not have to be disjointed. You can also find specializations in two large classifications. For example, hardware fault signals may be classified as power fault or backup fault, or some other fault. Even these may be further classified, such as storage, which is a form of data fault.

In this way, you can specify polymorphic events by modeling hierarchies of signals. Consider a state machine in which a transition is caused solely by the occurrence of a storage capacity. In this hierarchy, the transition can only be induced by the signal as a

leaf signal, such that it is not polymorphic. Consider a state machine with a transition due to a hardware fault. The transition is diversified that may be causing faults, such as Power fault, Storage fault, Display fault, and Backup fault.

Figure 4.5, depicts a series of signals that a Mobile Signal would be able to handle. Here it is worth noting that the root signal (Mobile Signal) is abstract, meaning that no direct instances are possible. This signal has two concrete specializations (Network and Hardware Fault), of which one is more specialized (Hardware Fault). The Network signal only has one parameter.

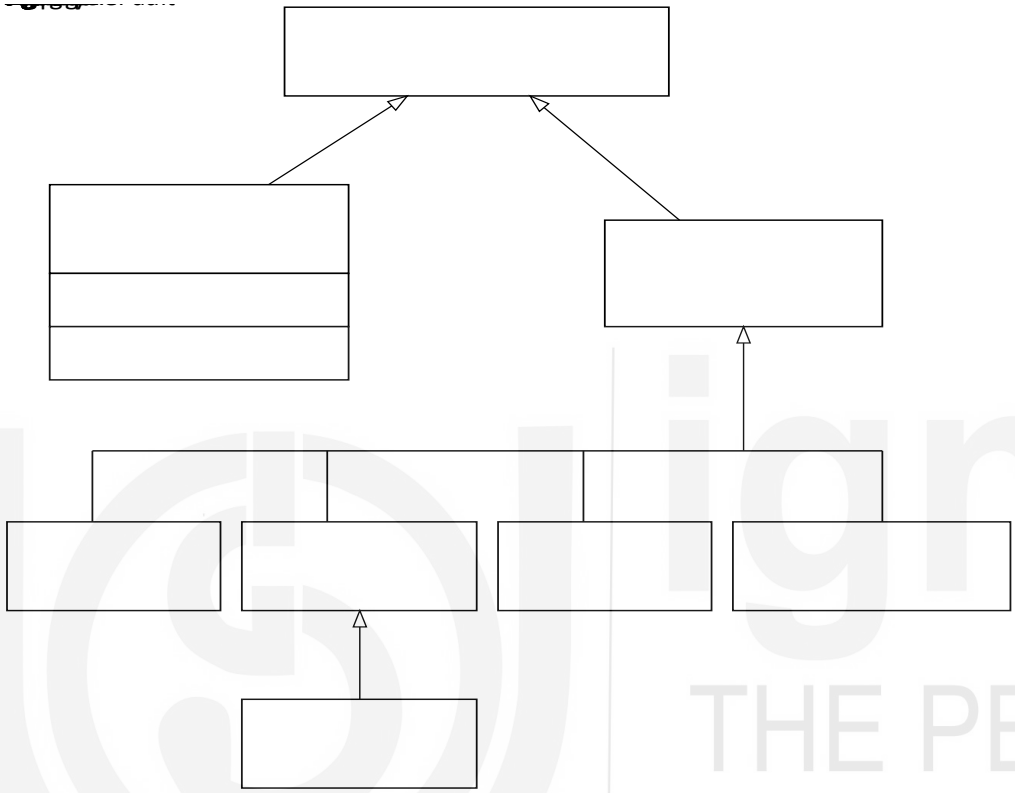

Figure 4.5: Modeling Families of Signals

- **Modeling Exceptions**

To model exceptions,

- Consider the extraordinary requirements that could be posed for each class, interface, and function of those components.
- Manage these exceptions in a logical order. Boost general, specialized, and if necessary, add intermediate exceptions.
- Define the exceptions that could be raised for any operation. This can be achieved directly (by displaying sending dependencies from an operation to the exceptions) or can be specified in operation.

A significant factor in modeling is visualizing the behavior of a class or interface, defining and reporting the exceptions to its operations. When a class or an interface is delivered, operations can call up will be simple, but there will be no clear exceptions for each operation unless you specifically model them.

Specification operations can contain exceptions. Exceptions are forms of signals in the UML that are modeled as stereotypes. Modeling exceptions behave in reverse of the modeling in the general signal family. A family of signals represented various types of

other signals that an active object can receive from model exceptions. A particular object can behave in general to define the type of exceptions.

In figure 4.6, a hierarchical structure of exceptions can be simply raised by a regular container type library like the template class. This hierarchy shows the abstract signal exception and consists of three highly specialized exceptions: doubling, overflow, and underflow. As illustrated, the Add() operation raises doubling and overflow exceptions, and the Delete() operation raises only the underflow exception. Additionally, you may have placed these dependencies in the background by naming them in each process specification. However, you can construct clients using the set class correctly by understanding which exceptions each procedure can send.

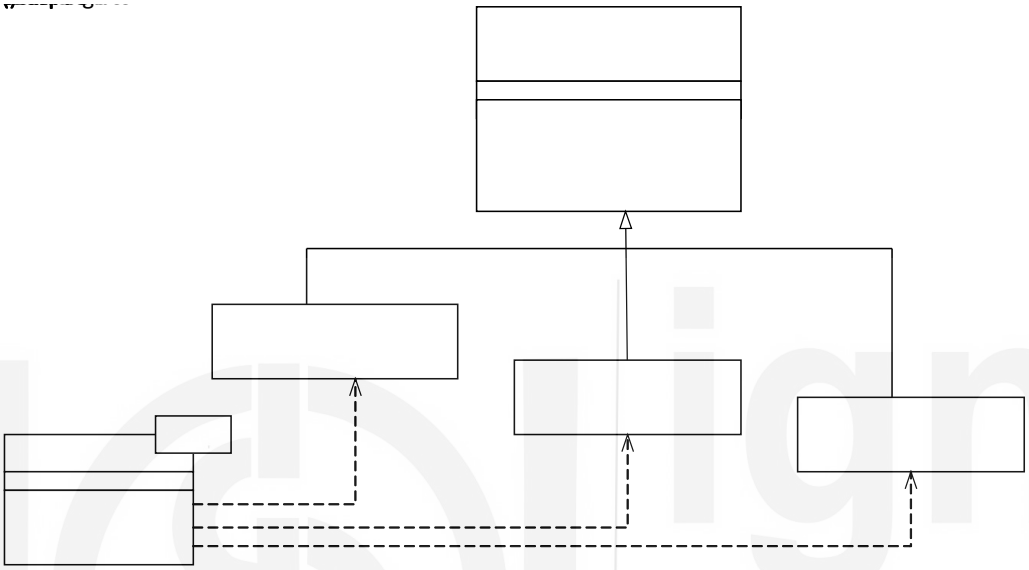

Figure 4.6: Modeling Exceptions

Check Your Progress 1

1) An event that represents the passage of time is _____.

(i) Event

(ii) Node

(iii) Time event

(iv) Signals

2) Time event is customized by the keyword_____.

(i) When

(ii) After

(iii) Signal

(iv) Change

3) Exceptions are arranged in _____.

(i) Hierarchy

(ii) Sequence

(iii) Linear

(iv) Circle

4) The change event is modeled by the keyword_____.

(i) After

(ii) When

(iii) Time

(iv) Signal

5) Define External and Internal Events.

……
……..……
……..

6) Explain how Sending and Receiving events are executed.

……
……..……
……..

4.4 STATE MACHINES

In UML modeling, state machine is a specification of the dynamic behavior of a particular class, objects, use-cases, and systems. When you create a state machine, the object attaches to the system to become the master of the state machine. When a state machine is used for operation, the operation becomes the master of the state machine. When you create a blank state machine diagram, it represents the sequence of state of an object, the events that cause a transition from one state to another state, and the result received from a machine is the change of the state. A blank state machine diagram is used to create a state machine. It is a graphical representation of the sequence of states of an object, which is used to make a transaction from one state to another state. Diagrams can add to describe the different behavior of the system. The state machine is also used to describe classes and systems that have substantial behavior. Diagrams of a state machine are used to describe different positions of the behavior of an object; if the object behavior is simple, that stores or retrieves the data. When the behavior of the other object is insignificant, then its state machine may be of no attraction to it. It can also store various states that represent different hierarchical state levels. Nested states are used to test the complex state changes in the objects. State machines are useful for modeling real-time or event-driven systems because they show the system's dynamic behavior. When we develop the state machine and its phases of a software project, then various situations are to be created, which are as follows:

4.3.1 State

The state is a circumstance in the course of an object's existence, during which some condition is fulfilled, some operation is carried out or some event is awaited. For a limited period of time, an object persists in the state. For example, in an ATM transaction, the ATM may be in either reading card state, reading PIN state, choosing transaction, or performing transaction.

- A state name should be unique in its neighboring state.
- A state contains five parts: name, input/outcome behavior internal transitions, and, transitions performed without any state change.
    - ***Substates*** – A state's nested configuration that includes disjoint or concurrent substates.
    - ***Deferred events*** - A collection of events that were not processed in this state, but in place of delayed object handling in a different state.

- ***Initial state*** – It shows the default starting point for the state machine or substate and is seen as a full black circle.
- ***The final state*** – In this state, machines or enclosures are closed and are shown as a filled black circle with an unfilled circle.
- **Pseudo-states -** The first and last states are known as pseudo-states.

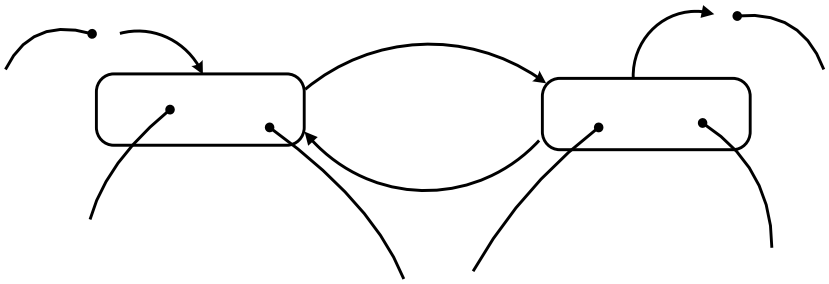

Figure 4.7: States of a System

4.3.2 Transitions

- Transition refers to the relation between two states; the first state performs some action and reaches the second state. Transactions are executed when one event is defined, and conditions are met.
- Transition fires mean state reform takes place. The entity will be executed before the transition fires. An object is said to be in the target state after the fires on it.

Five parts of a transition are:

- **Source state –** The transition's effect on the state.
- **Event trigger –** A stimulation that can activate a source state to fire when a guard condition is met.
- **Guard condition –** Boolean expression is evaluated when the transition is activated by the reception of the event trigger.
- **Action –** An executable atomic calculation that may be directly performed on the object that owns the state machine and indirectly on the other objects that are visible to the object.
- **Target state –** It is activated after the completion of the transition.

The transition has diverse sources and targets. Self-transition is a transition of equal source and destination states.

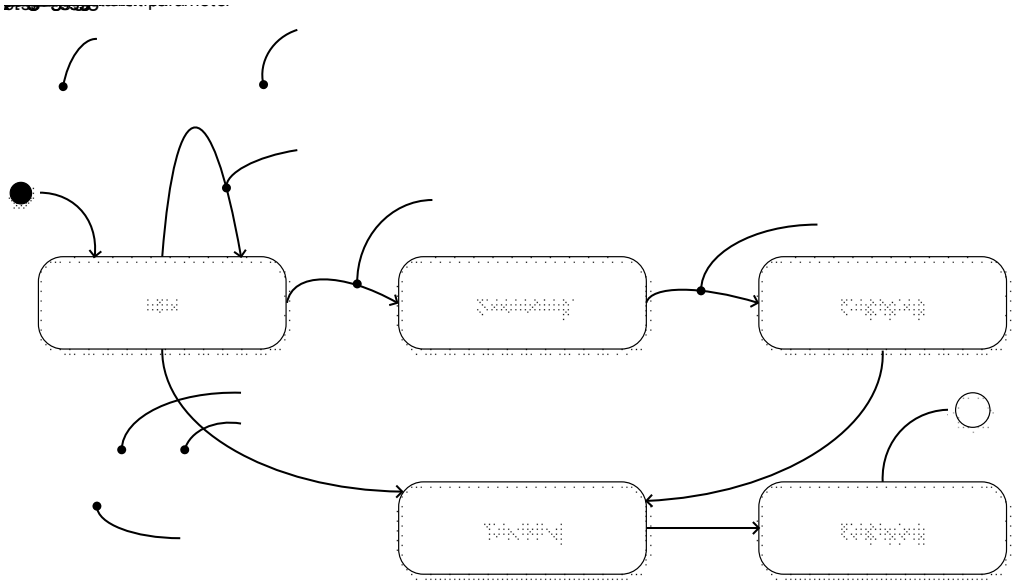

Figure 4.8: Parts of a Transition

4.3.3 Event Trigger

- In the context of state machines, an event is the presence of a signal that may trigger a state transition.
- Various changes of state can involve events like signals, calls, and time passing.
- An event – signal or call – can have transition parameters, including guard state and action expressions.
- The symmetric event trigger is also possible in a transition.

4.3.4 Guard Condition

- It is a Boolean function wrapped in square brackets after the trigger event.
- It is calculated after the transition's trigger event takes place.
- It is only calculated once for each transition as the event occurs, but it can be re-evaluated if the transition is reactivated.

4.3.5 Action

- An action is a discrete measurement that is executable, i.e. it cannot be disrupted by an occurrence and ends.
- Actions involve calling to work, making, removing, and transmitting a signal to an object.
- Other activities can cause an activity to be disrupted.

4.3.6 Advanced States and Transitions

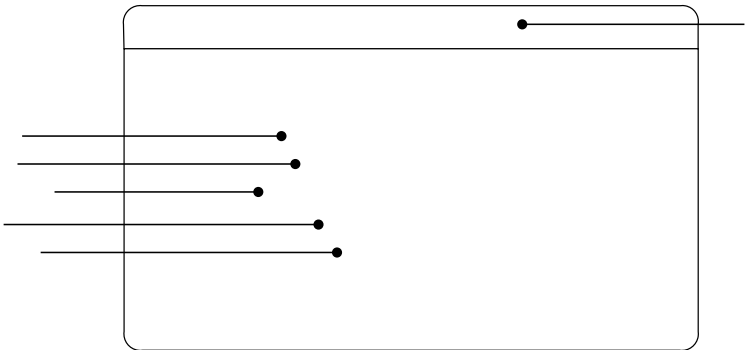

Figure 4. 9: Advanced States and Transitions

- **Entry and Exit Actions**

  - Entry activities are steps to be accomplished when entering in state and as specified by the keyword 'Initial' with the appropriate action.
  - Exit actions are actions to be performed while exiting a state marked with the Last' keyword event, along with appropriate action.

- **Internal Transitions**

  - An internal transition is an event that can be operated internally without the state leaving.
  - An internal transition can have parameter events and guard conditions.

- **Activities**

Activities use the idle time of an entity in a state. The 'Do' transition is used to determine the work to be performed within a state after sending the entry action.

- **Deferred Events**

A deferred event is a collection of events whose occurrence in the state is delayed until a state in which the mentioned events are not deferred becomes active, at which point they occur and can cause transitions as though they had just occurred. A deferred event is described by listing the event with the special action 'defer'.

4.3.7 Substates

- A substate is a state that is nested within another one.
- A composite state consists of substates.
- A composite state can contain either concurrent (orthogonal) or sequential (disjoint) sub-states.
- Substrates can be nested at any stage.

4.3.8 Sequential Substates

- Sequential substates are those substates in which each state can effectively exercise an occurrence common to the composite states at any time.
- Sequential substates split the composite state's, space into disjoint states.
- There can only be one initial state and one final state in a nested sequential state machine.

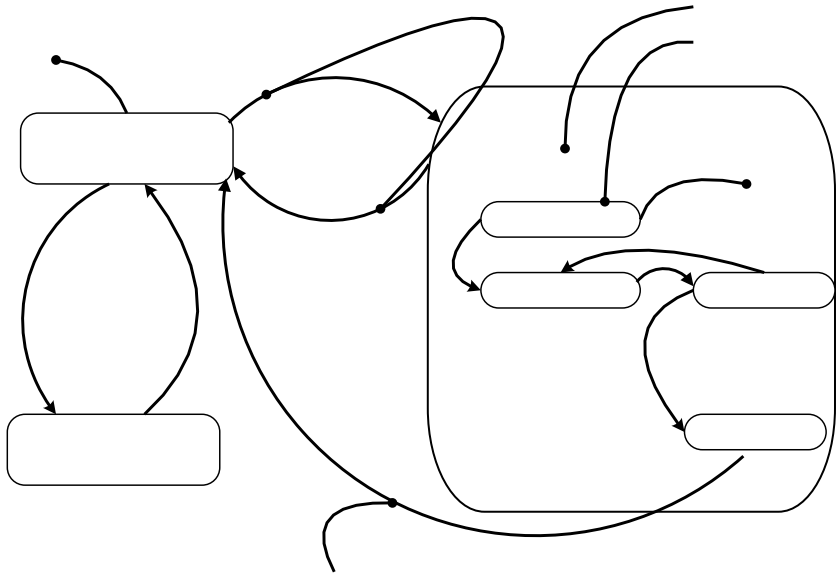

Figure 4.10: Sequential Substates

- **History States**

A history state requires a composite state of sequential substates to recall the last active substate before transitioning from that composite state. In figure 4.11 history state is shown. A small circle with the symbol H is depicted as a shallow history state.

- The composite state's first entry doesn't have history.
- The H symbol corresponds to a shallow history remembering only the history of the nesting state machine.
- The H* symbol designates a profound history that is symbolic of every depth of the nesting state.
- If only a stage of nesting is semi-equivalent to shallow and deep history.

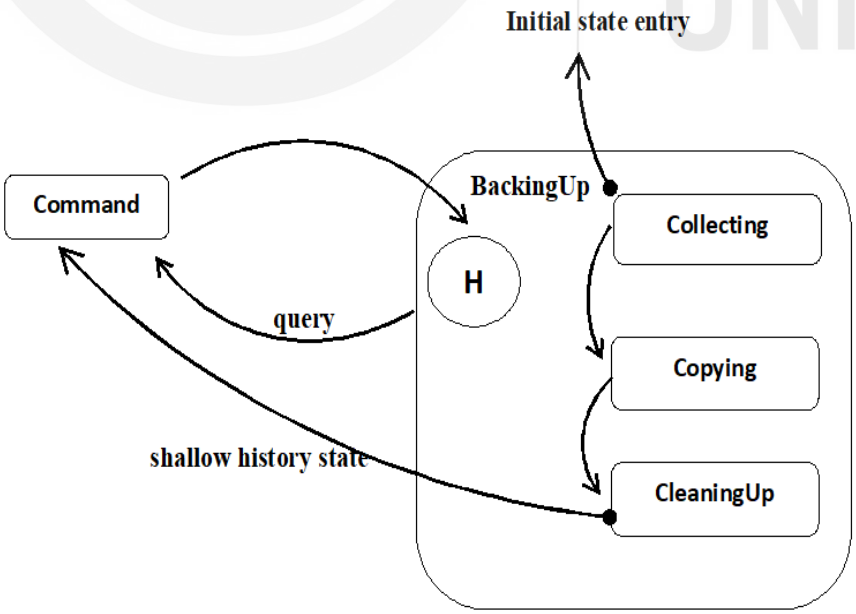

Figure 4.11: History State

4.3.9 Concurrent Substates

Concurrent substates specify two or three state machines operating in parallel in the enclosing object context. Execution of these concurrent substates continues in parallel. As shown in figure 4.12, these substances are waiting for one another to join in a single flow. Also, note that a nested concurrent state machine has no initial, final, or historical condition.

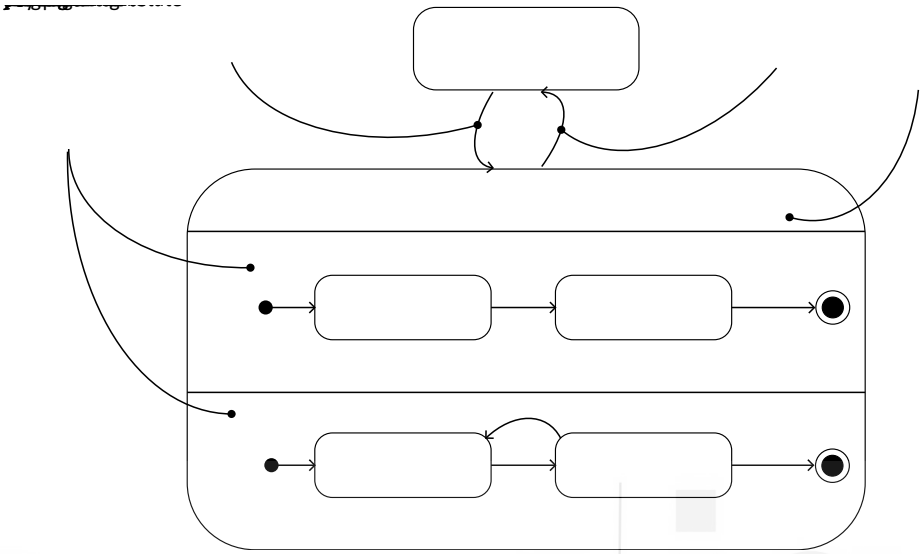

Figure 4.12: Concurrent Substates

4.3.10 Common Modeling Techniques

➢ **Modeling an Object's Lifetime:**

The next most traditional explanation for using state machines is to simulate an object's existence, in particular classes, case implementations, and the entire system. However, interactions shape the behavior of an artifact culture. A state machine forms the behavior of the actual entity over the lifespan, such as user interfaces, controls, and computers. When you model an object's lifespan, three aspects are specified:

1. what the object will respond to,
2. what happens, and
3. how past experiences influence current behavior.

Modeling an object's lifetime often entails agreeing on the sequence in which the object will meaningfully react to events, beginning with its creation and ending with its destruction.

To simulate an object's lifetime, you need to explain the framework for the state machine, whether it is a class, a use case, or the system as a whole.

- ➢ When the context is a class or a use case, get the nearby classes, such as the parents of the class and any classes available by associations or dependencies. These neighbors are potential targets for actions and should be included in guard conditions.

- ➢ When the context is the system as a whole, then limit the attention to one system behavior. Each entity in the system could engage in a system life-span model, and a full model will be intractable except for the most trivial systems.

Also, while simulating an object's lifetime followings should be considered:

- Set the original and final entity states. You should state the pre and post conditions of the initial and final states to lead the rest of your model.

- Determine the events to which this object will respond; if you have already defined these in the interfaces of the object. If you don't already define them, you would have to think about which objects in the setting would communicate with the object and which events they might display.

- From the initial to the final state, decide the top-level state in which the item can be placed. Link these states to the transitions induced by the subsequent events. Continue by adding actions to these transitions.

- Describe any initial or final actions, especially if the idiom they cover is being used in the state machine.

- Substates may be used to extend these states as needed.

- Make sure all events on the state machine follow the events predicted from the object's interface. Also, ensure that the state machine manages all events required from the object interface. Finally, search at places where you have clearly said that you chose to ignore events.

- Verify if an enclosing entity's situations, techniques, and processes support all the operations listed in the state machine. Detect through state machine manually or with software tools to equate sequences of events and their responses. Look for unreachable states where the machine can get stuck.

- After rearranging the state machine, double-check it against planned sequences to guarantee that you haven't altered the semantics of the object.

Figure 4.13, illustrates the state machine for the controller in a home security alert device, which is in charge of tracking numerous sensors around the house's perimeter.

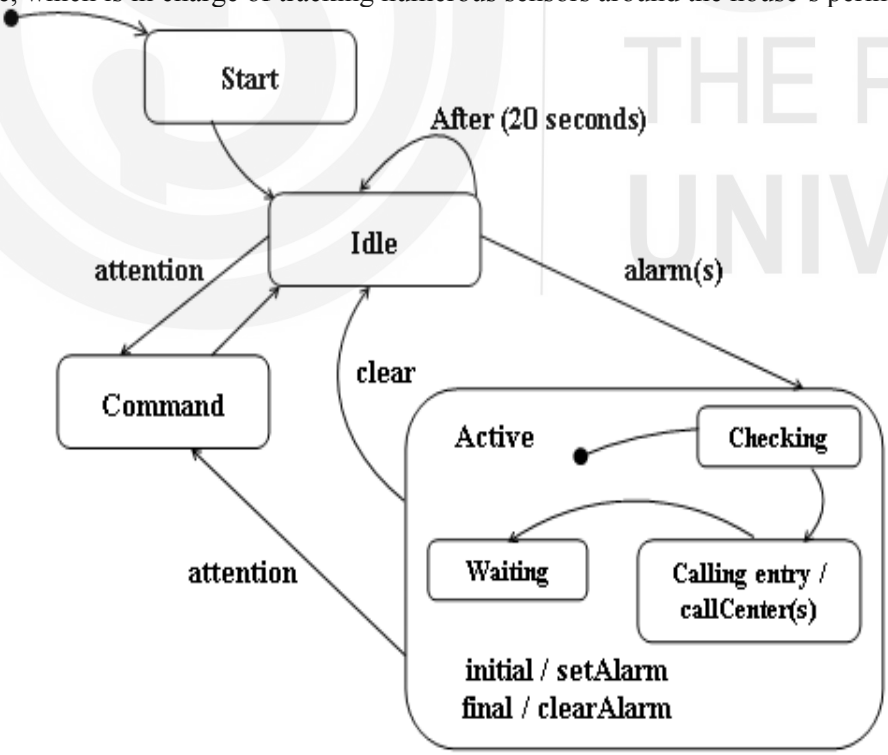

Figure 4.13: Modeling the Lifetime of An Object

Check Your Progress 2

1. A _____ is a relationship between two states indicating that an object in the first state will enter into the second state.

   a) Transition
   b) State
   c) Association
   d) Generalization

4. A state that has substates, known as nested state, is called_____.

   a) Composite state
   b) History state
   c) Target state
   d) Source state

5. Inside the state, the events are encountered to handle without leaving the state is known as _____.

   a) State machines
   b) State transaction
   c) Internal transaction
   d) External transaction

6. The relationship between two states is called_____.

   a) Transaction
   b) State
   c) Association
   d) Generalization

7. A guard condition is used to represent_____.

   a) Rectangle with rounded corners
   b) Dashed line
   c) Ellipse
   d) Square brackets

8. Define Events, States, and Transactions.

   ……

   ……..……

   ……..……

9. Draw a state machine diagram for Washing Machine.

   ……

   ……..……

   ……..……

4.5 PROCESSES AND THREADS

An active class is an object that has a process or thread. The class can monitor the operation of the process, while an active class is a class of instances. A process itself is a heavyweight flow that can run concurrently with many other processes. A thread is a lightweight flow that can run concurrently with several other threads in the same process. A class is represented by a rectangle of thick lines. Threads and processes are described as stereotyped active groups.

4.4.1 FLOW OF CONTROL

In a linear method, there is only one control flow, which means that only one thing will happen at a time. Whenever a sequential program is running, control begins at the beginning of the program, and procedures are executed in that order. If multiple users executed the concurrent programs, a sequential program would process only at a time. The queuing or dispatching of any concurrent external events is known as a flow of control. To track the operation of a sequential program and trace the source of execution flow from one instruction to the next, in sequential order. There is some recursion or loop in the program execution of loop, conditional, or jump statements, the flow of the expression returns to itself. In a sequential method, there can be only one execution flow. On the other hand, when we execute the program concurrently, there is more than one control flow, which means more than one statement is executed at a similar time. Using the concurrent system, multiple operations are performed, which are executed independently as a process or thread. In UML, active class is used to define a thread or a process, that is, the origin of the program or control flow.

4.4.2 CLASSES AND EVENTS

There are different types of classes: Active class and Normal class. Active classes have some special properties. It represents the flow of control while the Normal class has not any such flow control. However, active classes and normal classes behave implicitly called passive classes because they cannot control flow independently. When an active flow is created, the associated flow of control automatically starts; when the active object is deleted, all the associated flow of control is terminated itself.

The properties of active classes are the same as those of regular classes. Its instances are created. Attributes and operations are available in active classes. It is used to describe relationships of association, generalization, and connection. All of UML's configuration tools, such as stereotypes, tag attributes, and constraints, can be used for active classes. The identification of interfaces can be aided by active groups. Collaborations can be used to do this, and state machines can be used to describe the actions of an active class.

4.4.3 STANDARD ELEMENTS

The UML uses various standard elements, but most of the standard stereotypes which are applied to active classes are as follows:

1. Process

It is a heavy weight process, in which the operating system involves running more than one process at a time. These processes are run independently in a separate and autonomous environment. One process communicates with another process using communication techniques like signals, shared memory, semaphore files, etc. Processes are not nested with each other. If there are several processors on the node, then concurrent processes may execute at the same time.

2. Threads

It is a lightweight process in which threads execute inside a process and share the same address space with other threads of the process. In java programming, a thread is used as a class thread. All the threads in the process are connected to access the same resources inside the same process. Threads do not follow the nested concepts inside one another.

4.4.4 COMMUNICATION

When objects participate with another object by passing messages from one to the other, it is called communication between objects. There are four possible communications of interaction with passive and active objects are as follows:

- A signal can be transferred from one inactive object to the next. However, it is assumed that only one flow diagram can travel through these objects at a time.
- Second, a message may be sent from one working object to the next using intercrosses correspondence. There are two methods of communication.
  - An active object will synchronously invoke another operation.
  - An active object can send a signal or invoke an action of another object asynchronously.
- Third, a message should be sent from an active object to an inactive object.
- Fourth, a message can be transferred from a passive object to an active object.

4.4.5 SYNCHRONIZATION

Visualize the various control flows that are created temporarily through a concurrent system. If a flow goes through an operation, we state that the control locus is in operation at a specific time. We may also assume that since the operation is calculated for such classes, the control locus is in one particular class instance at any given moment. A single process (and hence in an object) can have multiple control flows, and different operations can have different control flows. But, in a single object, there are always many control flows.

The problem occurs when more than one control flow is concurrently executed in one entity. If there is no alert, one flow interferes with another, destroying the object's condition. This is the traditional issue of mutual exclusion. If failures occur and are not dealt with adequately, it may result in many race conditions and interference, allowing concurrent systems to fail in uncertain and unexpected conditions.

The trick to addressing this issue in object-oriented architectures is to consider an object as a critical area. To address this, there are three alternatives, each requiring the attachment of some synchronization features to the operations specified in a class. All methods can be modeled in the UML.

1. Sequential Model

As the object only has one flow at a time and the callers are synchronized beyond the object, the entity's semantics and consistency cannot be ensured in the face of multiple control flows.

2. Guarded Model

In the presence of several control flows, all calls to the guarded operations of the object shall be ensured to manage the semantics or dignity of the object. The object can be invoked for precisely one operation, reducing it to sequential semantics. The object's semantics and consistency are ensured by considering the process as atomic when there are many control flows.

3. Concurrent Model

Some programming languages support these concepts directly. For example, Java has

a sync property similar to the concurrent property of the UML. You may create support for all these properties in any language that encourages competitiveness through the construction of semaphores.

As illustrated in figure 4.14, you can associate these properties with an operation that can represent the UML with the use of constraint notation.

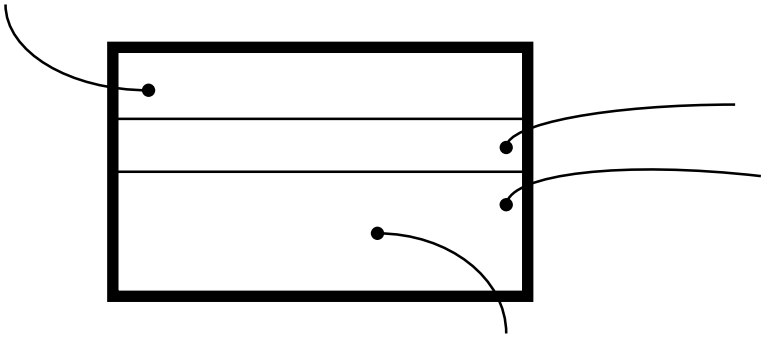

Figure 4.14: Synchronization

4.4.6 PROCESS VIEWS

In the process view, active objects play a significant role in imaging, assigning, designing, and validating a system. The system process covers all functions and threads that categorize the system performance and structure of synchronization.

4.4.7 COMMON MODELLING TECHNIQUES

➢ MODELING MULTIPLE FLOWS OF CONTROL

- Multiple control flows are complex to create. It is challenging to build a system that includes multiple control flows. It is difficult to determine how best to split the work into competitor active objects, but it is also difficult to formulate the appropriate coordination and alignment processes between active and passive objects on the scheme to ensure that they cooperate properly with these various flows. In the UML, you can do this by using class diagrams, as it helps to imagine the relationship between these flows.

- To understand the static semantics and relationship diagrams containing active classes and objects (to capture their dynamic explanations). To design multiple control flows, point out the possibilities for concurrent intervention, and conceptualize each flow as an active class. Create an active class by combining common sets of active objects. You have to take care that it does not go over the process view of your system by adding much more concurrency.

- Propose a structured separation of duties between such classes and then discuss the passive and active classes with each participant in the system. Ensure that every working class has the same range of characteristics, processes, and signals, and it is a closely unified and loosely coupled class.

- The diagram catches those static options in class, which highlights each active class.

- Consider the dynamic working of each class group.

- Ensure that the root of these flows is explicitly showing active objects. Identify each associated sequence with the active object name.

- Priority should be given to contact between active objects for synchronous and asynchronous messages.

- Priority should be given to synchronizing between these active and collaborative objects. Apply appropriate sequentially, guarded, or concurrent operation conceptually.

For example, figure 4.15, illustrates a portion of a trading system's mechanism view. SalesIndex, PriceWatcher, TradingManager, and NewsFeed (TV Newsfeed) are four items that simultaneously drive information through the system (named s, p, t, and c, respectively). Two of these objects (s and p) have their instances of the Analyst (a1 and a2). At least for this model, the Analyst can be constructed with the expectation that only one flow of control can be present in each of its instances at any given time. On the other hand, all analyst cases collaborate with an AlertManager (named m) at the same time.

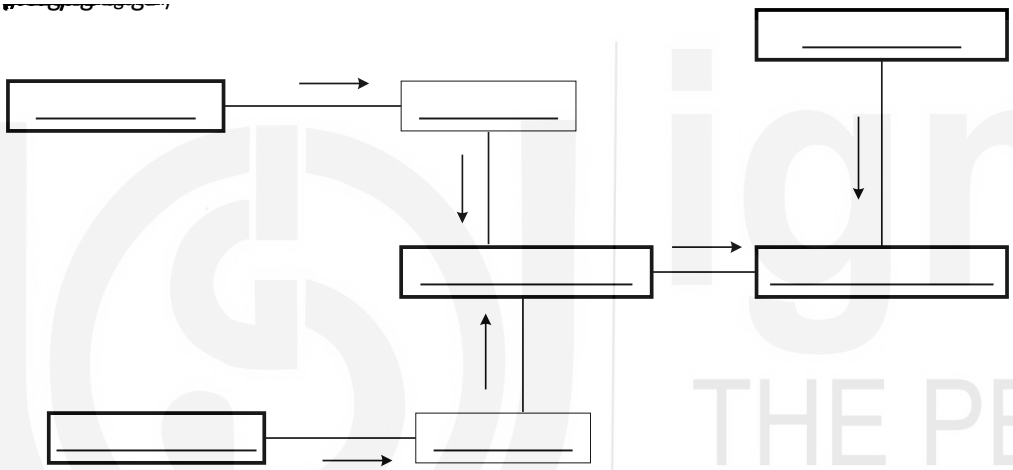

Figure 4.15: Modeling Flows of Control

Note:

Interaction diagrams are beneficial in visualizing when two control flows cross paths, and thus where the coordination and synchronization issues must be solved. In diagrams, the corresponding state machines are often usually attached, with orthogonal states which display the detail of each active entity. Also, tools may deliver many more distinct visual signals by coloring each flow.

➢ MODELLING INTERPROCESS COMMUNICATION

As part of integrating multiple control flows into the system, you will have to understand the methods used to communicate artifacts that exist in individual flows. Across threads (living in the same space), objects could interact through means of signals or calling events, in which the latter could display asynchronous or synchronous semantics. Over the processes, you typically need to use various structures. The inter-process connectivity issue is compounded because processes can exist on different nodes in distributed networks. Also, m***essage passing*** and ***Remote procedure calls (RPC)*** are the two other popular techniques for inter-process communication. In Unified Modeling Language (UML), these are often interpreted as

synchronous or asynchronous activities, respectively. To model inter-process communication:

- Simulate multiple control flows.
- Consider the processes serving these active objects and the threads.
- Use asynchronous communication to model sending messages; use synchronous communication to model remote procedure calls.
- Intimate the exact principle for communication by using notes, or more formally, by using collaborations.

A distributed reservation system with four nodes is illustrated in figure 4.16. The method stereotype is applied to each object. A location tagged value is often added to each object, indicating its physical location. Communication among the ReservationAgent, TicketingManager, and HotelAgent is asynchronous. It is defined as being built on a Java Beans messaging service, which is modeled after a node. The communication is synchronous with the Trip Planner and the Reservation System. The semanticity of your interaction is presented in the CORBA ORB collaboration. The Trip Planner acts as a client, and the ReservationAgent acts as a server.

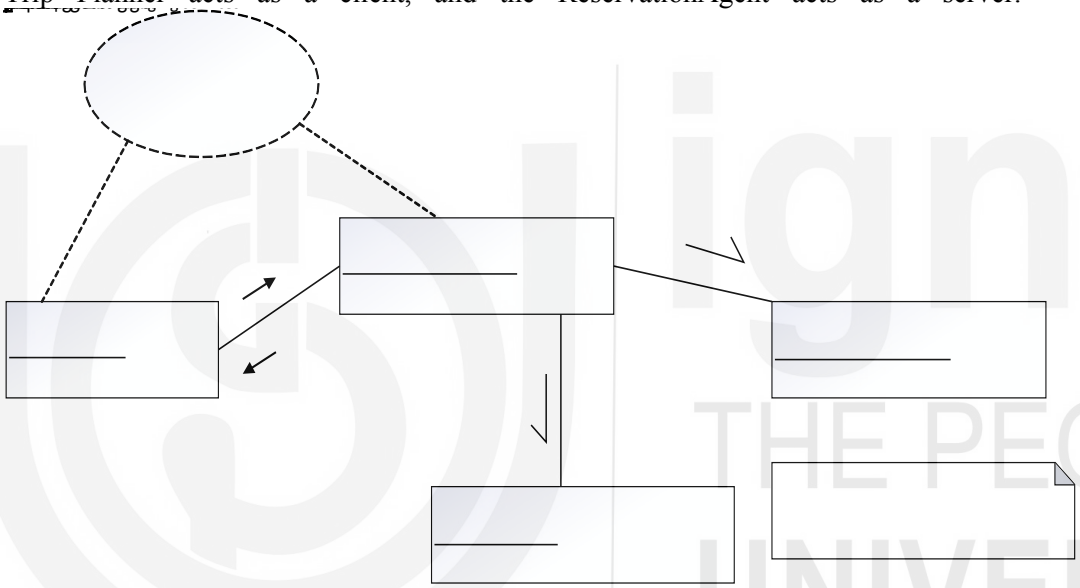

Figure 4.16: Modeling Interprocess Communication

4.6 TIME AND SPACE

In managing unpredictable incidents in which unique responses are needed at special times, there is a need for the management of time and systems resources. We understand that system resources have to be scattered across the globe, and some of these solutions will concentrate even more on increasing latency, synchronization, safety, and quality of service problems. A distributed framework requires the physical transfer of elements across nodes. These nodes may involve multiple physical processors in the same box or even represent computer systems located half a world away from each other.

The modeling of time and space is thus an integral part of a system in a distributed or real-time system. We use many UML features, including timing marks, time expressions, constraints, and tagged definitions, to visualize and document the corresponding building structures. It is really hard to have a good model dealing with these distributed and real-time processes, which exposes systems, time, space, and functionality.

Therefore, the UML provides a graphical representation of timing marks, time constraints, and location to cover real-time and distributed system modeling needs. Figure 4.17 shows the management of time and position.

A ***timing mark*** is a reference to the time an event takes place. The timing mark on the boundaries of the sequence diagram is represented graphically as a small hashmark (horizontal line).

A ***time expression*** is a concept that evaluates a time value, either absolute or relative. You may also use a message name to indicate a point of its processing, such as to request sendTime and request receiveTime.

A ***timing constraint*** is based on a relative or absolute time value semantic statement. Graphically, a time constraint in the form of a string of brackets is usually attached to an element by a dependency relation.

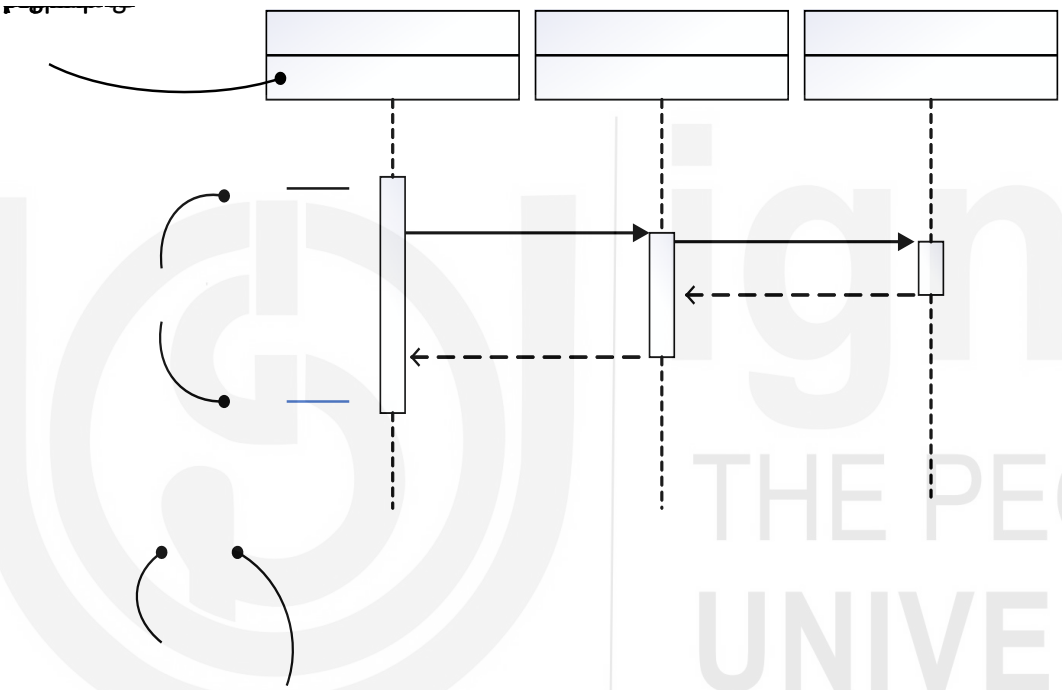

Figure 4.17: Timing Constraints and Position

4.5.1 Time

Real-time systems are time-critical systems. In such systems, events may arise at regular or irregular times. The reaction to the event itself may occur at pre-arranged absolute times or at predictable times. In figure 4.18, systems activity and associated time constraints are shown.

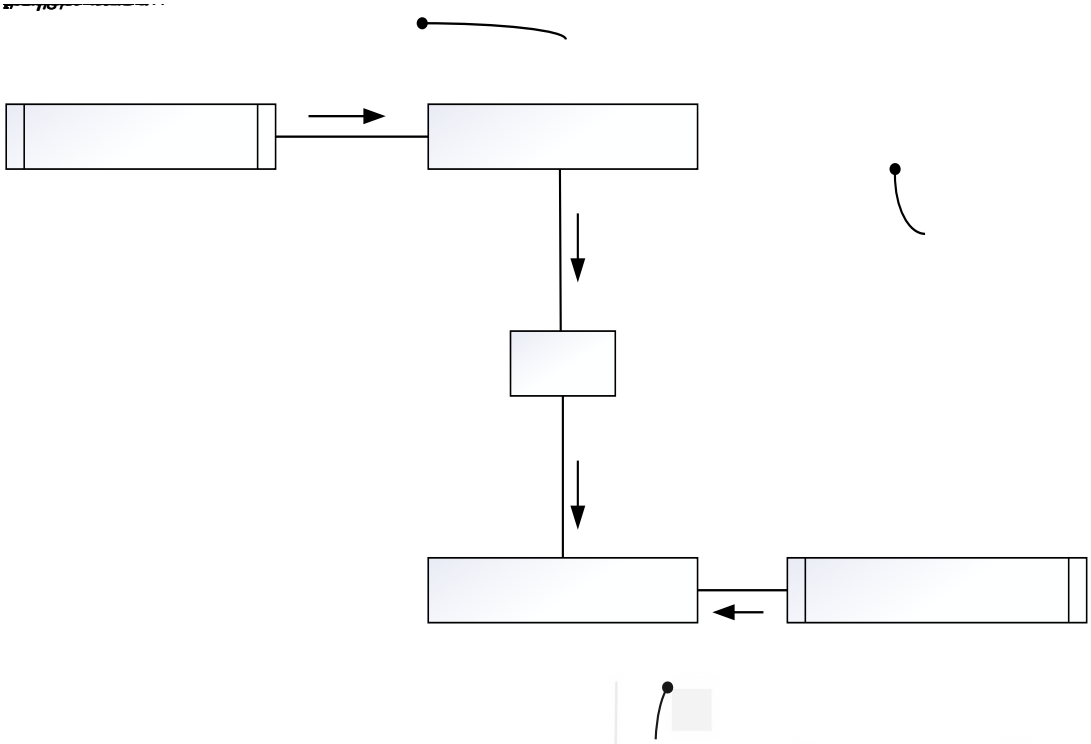

Figure 4.18: Time

4.5.2 Locations

In distributed systems, elements are physically distributed between the system nodes. For specific systems, components are set, while in some systems, which are loaded, components need to be migrated from one node to other nodes in other systems. Figure 4.19 shows different artifacts and their location by nesting, dependency, and implementations.

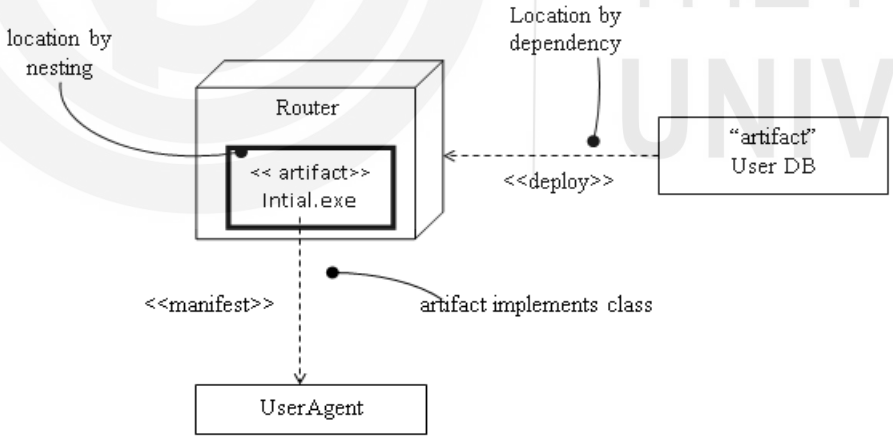

Figure 4.19: Location

4.5.3 Common Modeling Techniques

- **Modeling Timing Constraints**

To model timing constraints as illustrated in figure 4.20,

- Identify whether it needs to be set up at any absolute time for each event in a communication. You need to model the real-time property as a communication time limit if needed.

- A maximum relatively long time should be associated with each interesting sequence of messages during an interaction. Set a time limit for this real-time property.

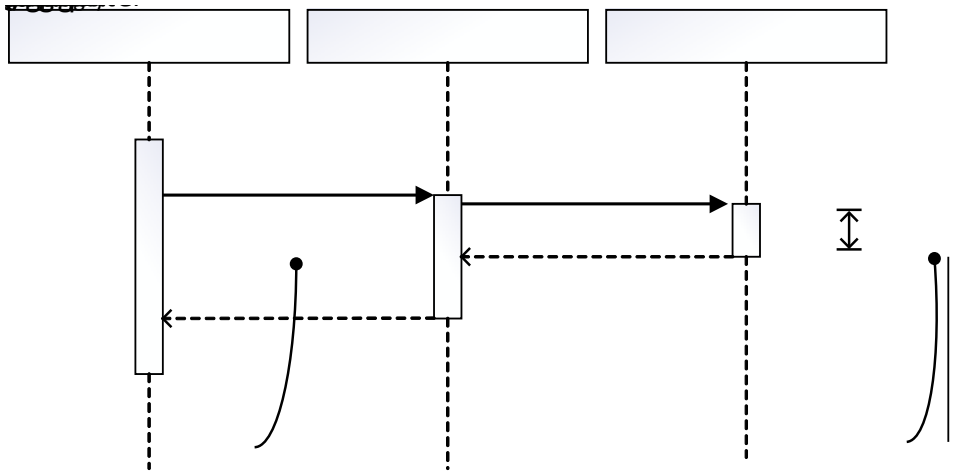

Figure 4.20: Modeling Timing Constraints

Proper distribution of objects is required in the system for proper functioning. Figure 4.21 shows a design of object distribution.

For object distribution following points are to be considered:

- Consider the object's location of reference with each significant class of objects throughout your system. All its neighbors and their states are taken into consideration. A closely linked location has nearby objects; a loosely connected location has far-off objects.
- Attribute objects closest to the actors, manipulating them tentatively.
- Consider interaction patterns among related objects.
- Object partition sets with low interaction degrees.
- Consider the system-wide allocation of obligations. To balance the load of every node, redistribute objects.
- Consider safety, volatility, and service quality problems as well, and redistribute the objects as necessary.
- Assign objects to modules, such that closely bound objects are on the same component.

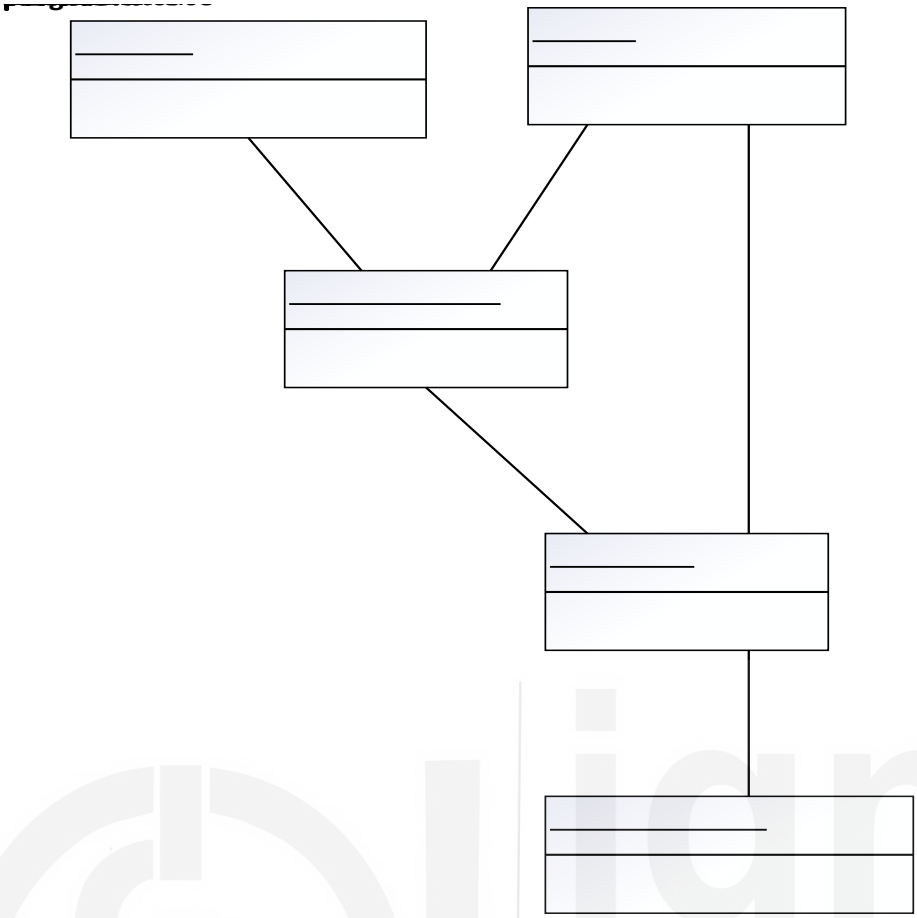

Figure 4.21: Modeling Distribution of Objects

Check Your Progress 3

1. Which diagram of UML is used to represent the time-ordering of messages?

a) Collaboration diagram

b) Sequence diagram

c) Class diagram

d) Activity diagram

2. What are the physical elements that exist at run time in UML?

a) Node

b) An Activity

c) An Interface

d) None of the above

3. The timing mark is represented by the……

a) Vertical Line

b) Dash Line

c) Horizontal Line

d) Dark Bold Line

4. Modeling of a time and space is represented by…….

a) Internal Event

b) External Event

c) Internal and external event

d) None of the above

5. What are the different types of communication between objects? Explain.

……

……..……

……..

6. What are the features of synchronization in UML? Explain.

……

……..……

…….

7. Draw a timing diagram for the execution of the virus in various states.

……

……..……

……..

4.7 STATECHART DIAGRAM IN UML

A Statechart diagram explains the different states of a component in a system. It can be used to define the different states of an object. External and internal events control these states.

- One of the essential diagrams in UML for modeling complex aspects of processes is the state diagram.
- It is used to present a state machine that displays the flow of control from one state to another state.
- It is essential for designing executable systems via forward and reverse engineering.
- It helps in modeling an object's lifespan.

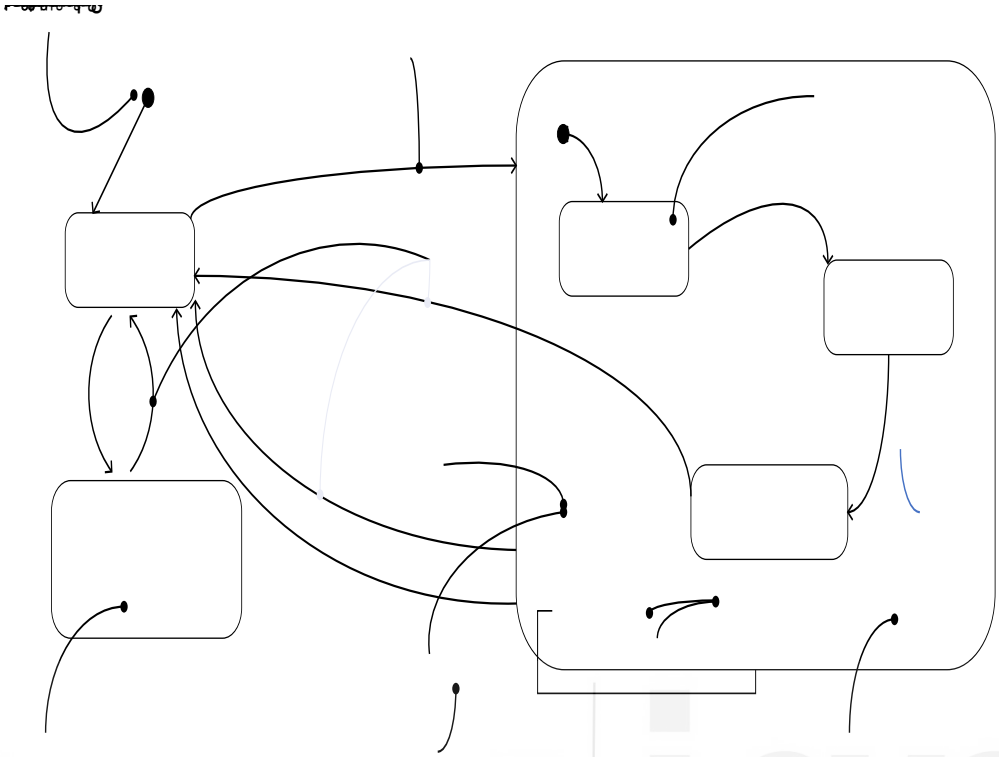

Figure 4.22: State Chart Diagram

As we can see in the above figure 4.22, a statechart diagram is nothing more than a schematic representation of a state machine. It emphasizes a flow of control from state to state.

4.6.1 Purpose of State Chart Diagram

It is one of the five UML diagrams. It is used to model the dynamic nature of a system. It describes various states of an entity and how it is modified by events. A reactive structure can be illustrated as a system that represents external or internal events. The statechart diagrams help model various reactive structures.

- The state diagram depicts the flow of control from one state to the next state. The term "state" refers to a situation in which there is an object, and it changes when some event is triggered. The main purpose of the statechart diagram is to design an object's lifespan from development to termination.

- Statechart diagrams are often required for forward and reverse engineering of a system. It is used to model the reactive system.

Below are the key goals of using state chart diagrams:

- To model a system's dynamic aspect.
- To model a reactive system's lifetime.
- To identify an object's different states throughout its existence.
- Define a state machine for modeling an object's states.

4.6.2 Terms and Concepts

Some terms and concepts used in statechart diagram are given below:

- A ***state*** is a condition in an object's life during which it fulfills a condition, executes an action, or waits for an event.

- A state machine is an ***event*** in the presence of a signal that may cause a state transition to occur.
- A ***transition*** is a relationship between two states. The first state performs certain actions and enters the second state after certain events occur and certain conditions are fulfilled.
- It is an ongoing non-atomic performance within a state machine. The behavior of a state machine is that object carries out while it is in a particular state. For example, when an account is in a particular state, the account holder's signature card is pulled. It also acts as an interruptible behavior.
- An ***entry action*** is a behavior that occurs while the object is transitioning into the state. Unlike an activity, an entry action is considered to be uninterruptible.
- An ***action*** is a discrete calculation, that is executable and results in a shift in the model state or the return of a value.
- An ***exit action*** occurs as part of the transition out of a state.
- An ***event-driven*** object is one, whose behavior is better specified by its response to events that occur outside of its context.

4.6.3 How to draw a State Chart Diagram

Knowledge of these object states is essential to analyze various objects and implement them correctly. A state chart diagram is used to illustrate the states of various objects.

Priority is placed on the changes of state in certain internal or external events as per requirements.

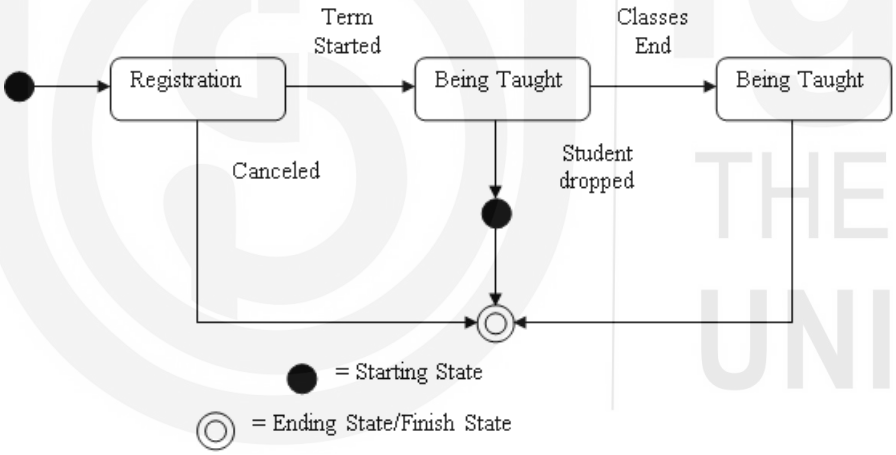

Figure 4.23: Student States in a Semester

To describe the states, statechart diagrams are very significant. States can be defined as the state of objects when a specific event occurs. Before drawing a statechart diagram, we should clarify the following points:

- Identify the significant objects to be examined.
- Define the states.
- Describe the events.

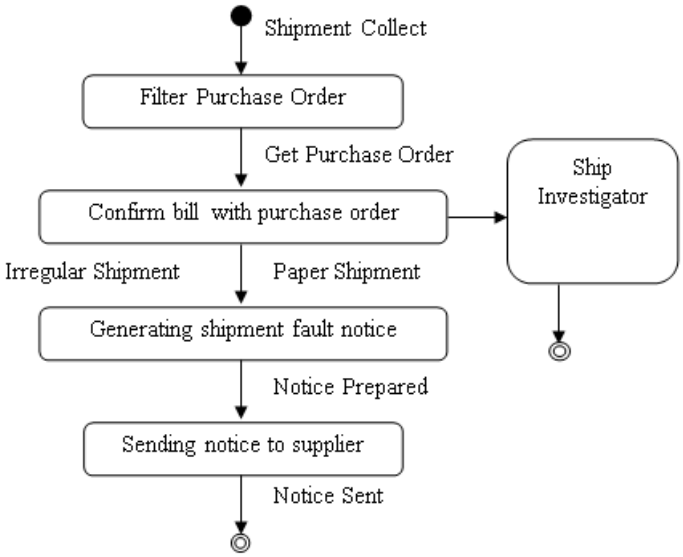

Figure 4.24: Shipping Process

In figure 4.23, the possible states of a student in a semester are shown. Also, a diagram of the shipping process is shown in figure 4. 24; referring to this diagram, one can easily interpret the shipping process activities.

4.6.4 Common Properties

Contents:

State diagrams usually contain basic states and composite states transition, events, and action. State diagrams can also contain notes and constraints.

Common Uses:

We use state diagrams to build a model of a structure's dynamic features. In the architecture of a system, these dynamic aspects can include the event-ordered behavior of any type of entity, including classes (which include active classes), interfaces, components, and nodes.

The state diagram may be concerning the whole structures, subsystems, or classes. We can also connect state diagrams to the use cases. In modeling the dynamic features of the system, class, or use case, a state diagram usually uses a reactive or event-driven object to model reactive objects, and the response to external events best marks its behavior. Thus, a reactive object is typically an ideal unit because its reaction depends on previous events when an event is received.

4.6.5 Common Modeling Process

The most common purpose for which we will use a state diagram is to model the behavior of the reactive objects.

Modeling Reactive Objects:

We essentially specify three things while modeling a reactive object's behavior:

- Stable conditions in which the object lives.
- Events that cause a state-to-state transition.
- The action is occurring in each change of state.

To model the reactive object, you have to:

- Choose the state machine context, be it a class, an application case, or a complete system.
- To guide the rest of your model, select the initial and final state for the objects, and indicate the possible pre and post conditions of both initial and final states.
- Decide on the object's stable states, taking into consideration the conditions under which the objects may exist during certain identifiable times. Start with the objects' high state and only then take into consideration of their possible sub-states.
- Decide the meaningful partial arrangement over the object's lifetime for stable states.
- Determine which events can trigger a state transition. Design such events as triggers for transitions to another state from one legal framework.
- Add actions to these kinds of transitions, such as on a Mealy machine or these states as in Moore machine.
- Remember how a machine can be simplified by using sub-states, branches, forks, joins, and history.
- Verify that it is possible to reach all states under certain event combinations.
- Check that there is no dead-end for any combination of events to exit the object.
- Track the state machine manually or with tools to search it for expected event sequences and responses.
- 'Tag' is the first string, and a message body is the second string.

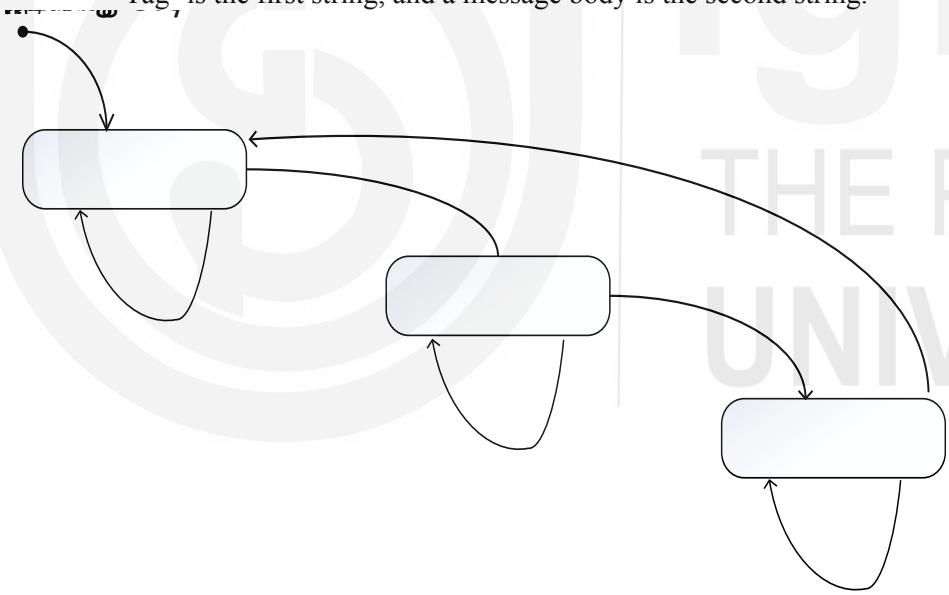

Figure 4.25: Modeling Reactive Objects

Figure 4.25, shows the modeling of reactive objects; it depicts the state of forwarding a basic context-free language to XML, such as that used in a system that streams in or out messages. Consequently, the machine is built as a parser (stream of characters) which matches the syntax in this case.

State Chart Diagram Case Study:
Let us try to understand the statechart diagram with the help of an example in which the state of the Order object is analyzed, which are as follows:

- An idle state is the first stage of the process. With the activities such as

submitting requests, verifying the request, and dispatching instructions, we will be arriving at the next states. These events are responsible for modifying the state of the command object.

- An object(object order) goes through the following stages during its life cycle, and also abnormal exits are possible. Due to some device issues, this abnormal exit may happen. Completing the whole life cycle is known as a complete transaction, as shown in figure 4.26. In the following diagram, the initial and final state of an entity also has been shown.

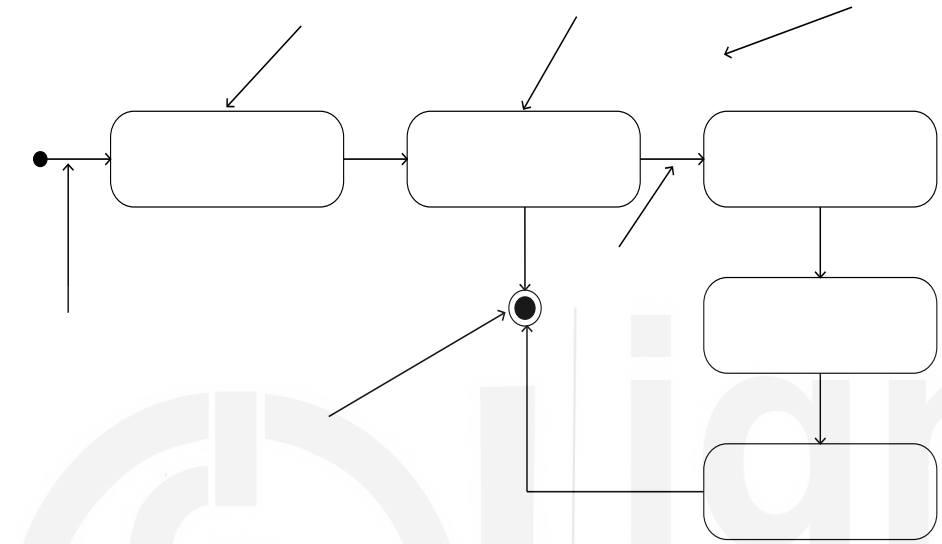

Figure 4.26: State Chart Diagram of an Order Management System

4.6.6 Where to Use State Chart Diagrams

The dynamical aspect of a system is modeled with state chart diagrams. However, it has several distinctive features for dynamic nature modeling. Its purpose is to explain the changes of state that arise as a result of events. It describes the component states, and these changes in the state are dynamic. Events have an internal or external effect on the mechanism.

Statechart diagrams are used to simulate the system's states as well as the events that occur. State map diagrams are used to clarify various states of an entity over its lifespan, which is very useful when developing a system. When these states and events are defined, they are used to construct a model, which is then used during the system's implementation.

When we look at how the statechart diagram is used in nature, we can see that it is specifically used to examine the object states that are affected by events.

The common use of statechart diagram is:

- Model system object states.
- The reactive framework can be modeled. The reaction mechanism is made up of reactive objects.
- To recognize events for improvements to the state.
- For reverse and forward engineering.

☞Check Your Progress 4

1. The element of state chart diagrams with round corner boxes to _____

a) Lifeline marker
b) Iteration marker
c) Transactions
d) State

2. The elements in state chart diagrams are_____

a) Transactions
b) Condition makers
c) Iteration makers
d) Lifeline makers

3. A solid circle with an outgoing arrow is represented by the _____

a) Final state
b) Zero-degree state
c) Initial state
d) Two-degree state

4. Which of the following statement is correct?

a) A transaction may happen all sudden, but usually, some events trigger them.
b) An event is interesting at a particular time; when events have no duration.
c) A transaction is a switch from one state to another state.
d) All of them

5. What are the different elements of a state chart diagram? Explain briefly.

……
……
……

6. Draw a state chart diagram for Library Management System.

……
……
……

4.8 SUMMARY

Most of the events in any system happened based on events, time, and signals. The structure and its relationship define the class model of a particular time. An event is a model specification that takes place in a particular time and space. When objects are attached to the system to perform various operations behave as the master of the state machine. The state machine is used to represent in the form of graphical representation to make a transaction from one state to another state. Active class is used to monitor the instances of the class while threads and processes are used to monitor the operations of the process. UML provides many other features like timing marks, time expressions, constraints, etc. to visualize the structure of the system. Different components of a system are described statechart diagram and describe the properties to use the diagram.

In this unit, basic norms, concepts, and style guidelines for constructing behavioral or dynamic UML models are explained. Unit explained sequence, collaboration, state and activity diagrams, and their elements. Behavioral modeling is the modeling of a system's behavioral or dynamic component. Further, this unit explains the interaction between the components that make up the system functionality, and it's used to identify how a system meets its requirements, parts, and relationships in a specification model for the system, subsystem, and class specification. Also, it is explained that the class, object, sequence, collaboration, state, activity diagrams, and their model components make up a specification model.

4.9 SOLUTIONS / ANSWER TO CHECK YOUR PROGRESS

CHECK YOUR PROGRESS 1

1. **C**
2. **B**
3. **A**
4. **B**
5. An external event is also known as a system event, for example, the event which is happened outside the boundary like an actor. For example, when a cashier presses the "List of Item" button on a POS computer.

   An internal event happened inside the system boundary. Internal events invoked a message or signal sent from another internal object. For example, when a sale receives a list of item messages from another system, it occurs as an internal event.
6. Two objects are executed in sending and receiving events: a sender and a receiver. When the sender sends the signals to the receiver end, it continues to dispatch the signals without waiting for a return from the receiver ends. In contrast, this operation performs the operation in which the sender waits for the receiver to respond before resuming its flow of control.

CHECK YOUR PROGRESS 2

1. **A**
2. **A**
3. **C**
4. **A**
5. **D**
6. An event is a significant or remarkable occurrence of an event like a telephone receiver hanging off the hook.

   A state is the condition of an object that happened between two events

like a telephone is idle after the receiver is placed on the bent, till the next call.

A transaction is a relationship between two states which happens from one state to another state, like when the telephone transaction occurs from an idle state to another active state.

7. In this state machine diagram, various state of the washing machine is represented. When the washing machine starts, "Washing" from rising to spinning is represented.

   If a power cut occurs, the washing machine will stop running and will hold at the "Power Off" state. When power is "On" the running state is entered at the "History State". It should resume from where it is started.

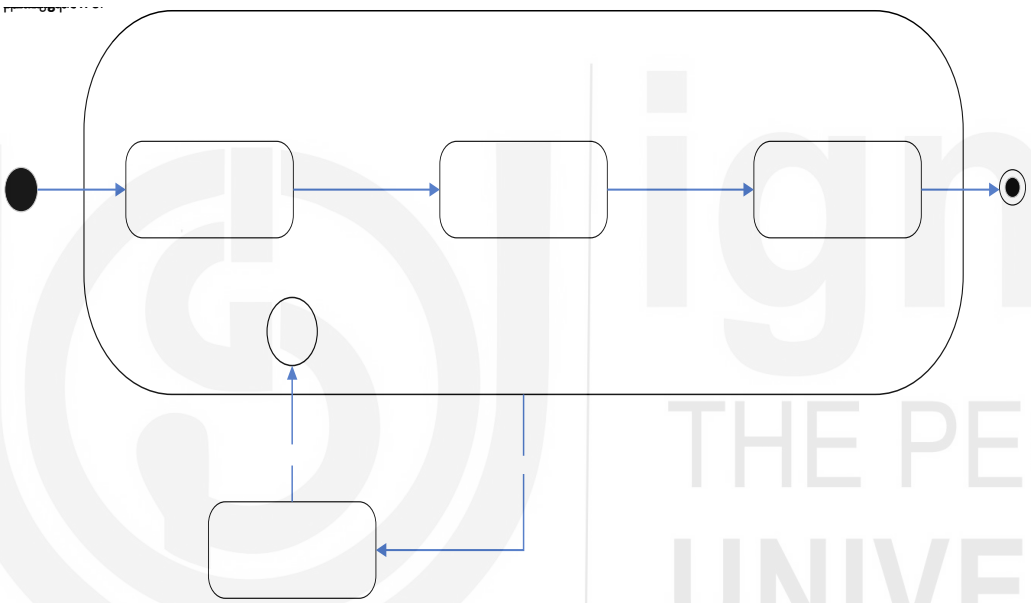

Figure 4.27: State Machine Diagram-Washing Machine State

CHECK YOUR PROGRESS 3

1. **B**
2. **A**
3. **C**
4. **A**
5. **There are four types of communication with passive and active objects which are as follows:**

- A signal can be transferred from one object to another object. But only one flow diagram can travel at a time.
- A message may be sent with two methods of communication synchronously and asynchronously.
- Multiple messages may be sent from one active object to one active

object of flow control.

- A message may be transferred from the passive object to the active object if all the flow control is executed in the active object.

6. **There are three features to handle synchronization, which are as follows:**

- Sequential: Only one flow is allowed in the object at a time.
- Guarded: Multiple control flows are allowed in the object using the guarded operations.
- Concurrent: Multiple flows of control are guaranteed by treating each operation as atomic.

7. **Timeline of Virus Execution**

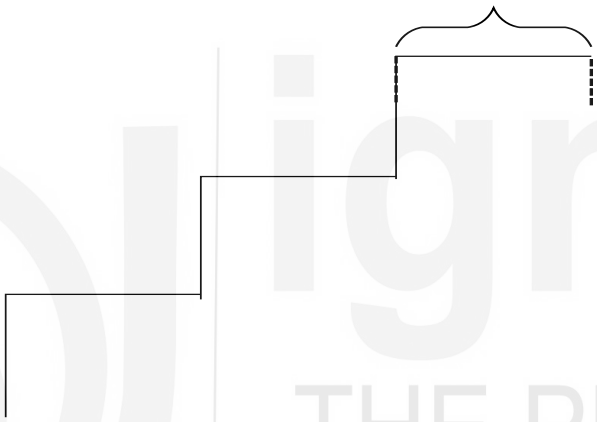

Figure 4.28: Timeline of Virus Execution

CHECK YOUR PROGRESS 4

1. **D**
2. **A**
3. **C**
4. **D**
5. **Elements of a state chart diagram are as follows:**
    - Initial state
    - State
    - Transaction
    - Internal transaction
    - Mutation event
    - Action
    - Guard condition

- Final state

6. Draw a state chart Diagram for Library Management System.

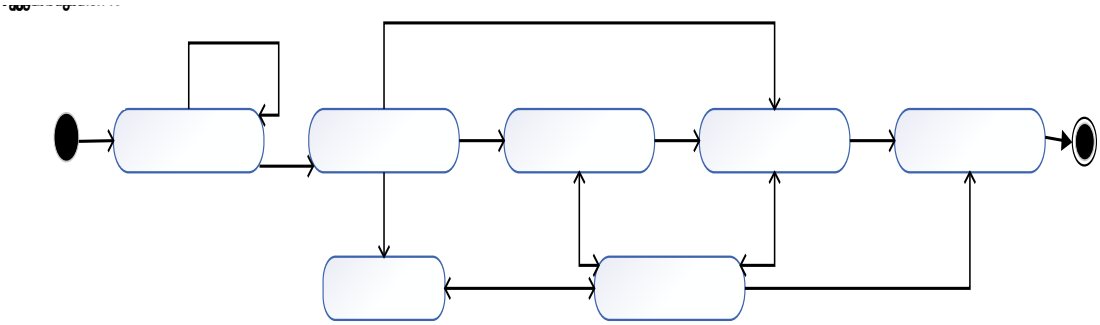

Figure 4.29: State Chart Diagram for Library Management System

4.9 REFERENCES/FURTHER READING

- Grady Booch, James Rumbaugh, and Ivar Jacobson,“ The Unified Modeling Language User Guide”, 2nd Edition, Addison-Wesley Object Technology Series, 2005.
- Grady Booch, Robert A. Maksimchuk, Michael W. Engle, Bobbi J. Young, Jim Conallen, Kelli A. Houston, “Object-Oriented Analysis and Design with Applications,” 3rd Edition, Addison-Wesley, 2007.
- James Rumbaugh, Ivar Jacobson, and Grady Booch, “Unified Modeling Language Reference Manual,” 2nd Edition, Addison-Wesley Professional, 2004.
- John W. Satzinger, Robert B. Jackson, and Stephen D. Burd, “Object-oriented analysis and design with the Unified Process,” 1st Edition, Cengage Learning India, 2007.
- Brett McLaughlin, Gary Pollice, and Dave West, “Head First Object-Oriented Analysis and Design: A Brain-Friendly Guide to OOA&D,” Shroff Publisher, First edition,2006.