

Experiment 05- Implement State/Activity diagrams

Learning Objective: To implement dynamic view of a system using Activity/State diagrams.

Tools: MS Word, draw.io

Theory:

Capturing the dynamic view of a system is very important for a developer to develop the logic for a system. State chart diagrams and activity diagrams are two popular UML diagram to visualize the dynamic behavior of an information system.

In this experiment, we will learn about the different components of activity diagram and state chart diagram and how these can be used to represent the dynamic nature of an information system.

State-chart Diagrams

In case of Object Oriented Analysis and Design, a system is often abstracted by one or more classes with some well defined behavior and states. A *state chart diagram* is a pictorial representation of such a system, with all its states, and different events that lead transition from one state to another.

To illustrate this, consider a computer. Some possible states that it could have are: running, shutdown, hibernate. A transition from running state to shutdown state occur when user presses the "Power off" switch, or clicks on the "Shut down" button as displayed by the OS. Here, clicking on the shutdown button, or pressing the power off switch act as external events causing the transition.

State-chart diagrams are normally drawn to model the behavior of a complex system. For simple systems this is optional.

Building Blocks of a State-chart Diagram

State

A state is any "distinct" stage that an object (system) passes through in it's lifetime. An object remains in a given state for finite time until "something" happens, which makes it to move to another state. All such states can be broadly categorized into following three types:

- Initial: The state in which an object remain when created
- Final: The state from which an object do not move to any other state [optional]
- Intermediate: Any state, which is neither initial, nor final

As shown in figure-01, an initial state is represented by a circle filled with black. An intermediate state is depicted by a rectangle with rounded corners. A final state is represented by a unfilled circle with an inner black-filled circle.

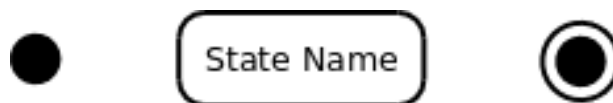


Figure-01: Representation of initial, intermediate, and final states of a state chart diagram

Intermediate states usually have two compartments, separated by a horizontal line, called the name compartment and internal transitions compartment. They are described below:

- Name compartment: Contains the name of the state, which is a short, simple, descriptive string
- Internal transitions compartment: Contains a list of internal activities performed as long as the system is in this state

The internal activities are indicated using the following syntax: action-label / action-expression. Action labels could be any condition indicator. There are, however, four special action labels:

- **Entry:** Indicates activity performed when the system enter this state
- **Exit:** Indicates activity performed when the system exits this state
- **Do:** indicate any activity that is performed while the system remain in this state or until the action expression results in a completed computation
- **Include:** Indicates invocation of a sub-machine

Any other action label identifies the event (internal transition) as a result of which the corresponding action is triggered. Internal transition is almost similar to self transition, except that the former doesn't result in execution of entry and exit actions. That is, system doesn't exit or re-enter that state. Figure-02 shows the syntax for representing a typical (intermediate) state

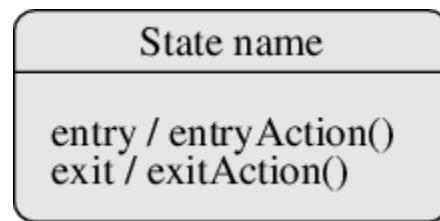


Figure-02: A typical state in a state chart diagram

States could again be either simple or composite. Here, however, we will deal only with simple states.

Transition

Transition is movement from one state to another state in response to an external stimulus (or any internal event). A transition is represented by a solid arrow from the current state to the next state. It is labeled by: event [guard-condition]/[action-expression], where

- **Event** is the what is causing the concerned transition (mandatory) -- Written in past tense
- **Guard-condition** is (are) precondition(s), which must be true for the transition to happen
- **Action-expression** indicate action(s) to be performed as a result of the transition

It may be noted that if a transition is triggered with one or more guard-condition(s), which evaluate to false, the system will continue to stay in the present state. Also, not all transitions do result in a state change. For example, if a queue is full, any further attempt to append will fail until the delete method is invoked at least once. Thus, state of the queue doesn't change in this duration.

Action

An action represents behavior of the system. While the system is performing any action for the current event, it doesn't accept or process any new event. The order, in which different actions are executed, is given below:

1. Exit actions of the present state
2. Actions specified for the transition
3. Entry actions of the next state

Activity Diagrams

Activity diagrams fall under the category of behavioral diagrams in Unified Modeling Language. It is a high level diagram used to visually represent the flow of control in a system. It has similarities with traditional flow charts. However, it is more powerful than a simple flow chart since it can represent various other concepts like concurrent activities, their joining, and so on.

Activity diagrams, however, cannot depict the message passing among related objects. As such, it can't be directly translated into code. These kinds of diagrams are suitable for confirming the logic to be implemented with the business users. These diagrams are typically used when the business logic is complex. In simple scenarios it can be avoided entirely.

Components of an Activity Diagram

Below we describe the building blocks of an activity diagram.

Activity

An activity denotes a particular action taken in the logical flow of control. This could simply be invocation of a mathematical function, alter an object's properties and so on. An activity is represented with a rounded rectangle, as shown in table-01. A label inside the rectangle identifies the corresponding activity.

There are two special types of activity nodes: initial and final. They are represented with a filled circle, and a filled in circle with a border respectively (table-01). Initial node represents the starting point of a flow in an activity diagram. There could be multiple initial nodes, which mean that invoking that particular activity diagram would initiate multiple flows.

A final node represents the end point of all activities. Like an initial node, there could be multiple final nodes. Any transition reaching a final node would stop all activities.

Flow

A flow (also termed as edge or transition) is represented with a directed arrow. This is used to depict transfer of control from one activity to another, or to other types of components, as we will see below. A flow is often accompanied with a label, called the guard condition, indicating the necessary condition for the transition to happen. The syntax to depict it is [guard condition].

Decision

A decision node, represented with a diamond, is a point where a single flow enters and two or more flows leave. The control flow can follow only one of the outgoing paths. The outgoing edges often have guard conditions indicating true-false or if-then-else conditions. However, they can be omitted in obvious cases. The input edge could also have guard conditions. Alternately, a note can be attached to the decision node indicating the condition to be tested.

Merge

This is represented with a diamond shape, with two or more flows entering, and a single flow leaving out. A merge node represents the point where at least a single control should reach before further processing could continue.

Fork

Fork is a point where parallel activities begin. For example, when a student has been registered with a college, he can in parallel apply for student ID card and library card. A fork is graphically depicted with a black bar, with a single flow entering and multiple flows leaving out.

Join

A join is depicted with a black bar, with multiple input flows, but a single output flow. Physically it represents the synchronization of all concurrent activities. Unlike a merge, in case of a join all of the incoming controls **must be completed** before any further progress could be made. For example, a sales order is closed only when the customer has received the product, **and** the sales company has received its payment.

Note

UML allows attaching a note to different components of a diagram to present some textual information. The information could simply be a comment or may be some constraint. A note can be attached to a decision point, for example, to indicate the branching criteria.

Partition

Different components of an activity diagram can be logically grouped into different areas, called partitions or swim lanes. They often correspond to different units of an organization or different actors. The drawing area can be partitioned into multiple compartments using vertical (or horizontal) parallel lines. Partitions in an activity diagram are not mandatory. The following table shows commonly used components with a typical activity diagram.



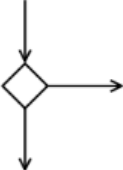
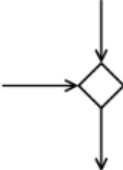
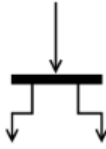
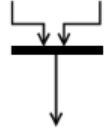

Component	Graphical Notation
Activity	
Flow	
Decision	
Merge	
Fork	
Join	
Note	

Table-01: Typical components used in an activity diagram

A Simple Example

Figure-04 shows a simple activity diagram with two activities. The figure depicts two stages of a form submission. At first a form is filled up with relevant and correct information. Once it is verified that there is no error in the form, it is then submitted. The two other symbols shown in the figure are the initial node (dark filled circle), and final node (outer hollow circle with inner filled circle). It may be noted that there could be zero or more final node(s) in an activity diagram.

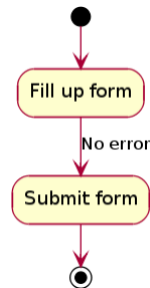


Figure-04: A simple activity diagram.

Procedure:

Guidelines for drawing State chart Diagrams

Following steps could be followed, to draw a state chart diagram:

- For the system to developed, identify the distinct states that it passes through
- Identify the events (and any precondition) that cause the state transitions. Often these would be the methods of a class as identified in a class diagram.
- Identify what activities are performed while the system remains in a given state

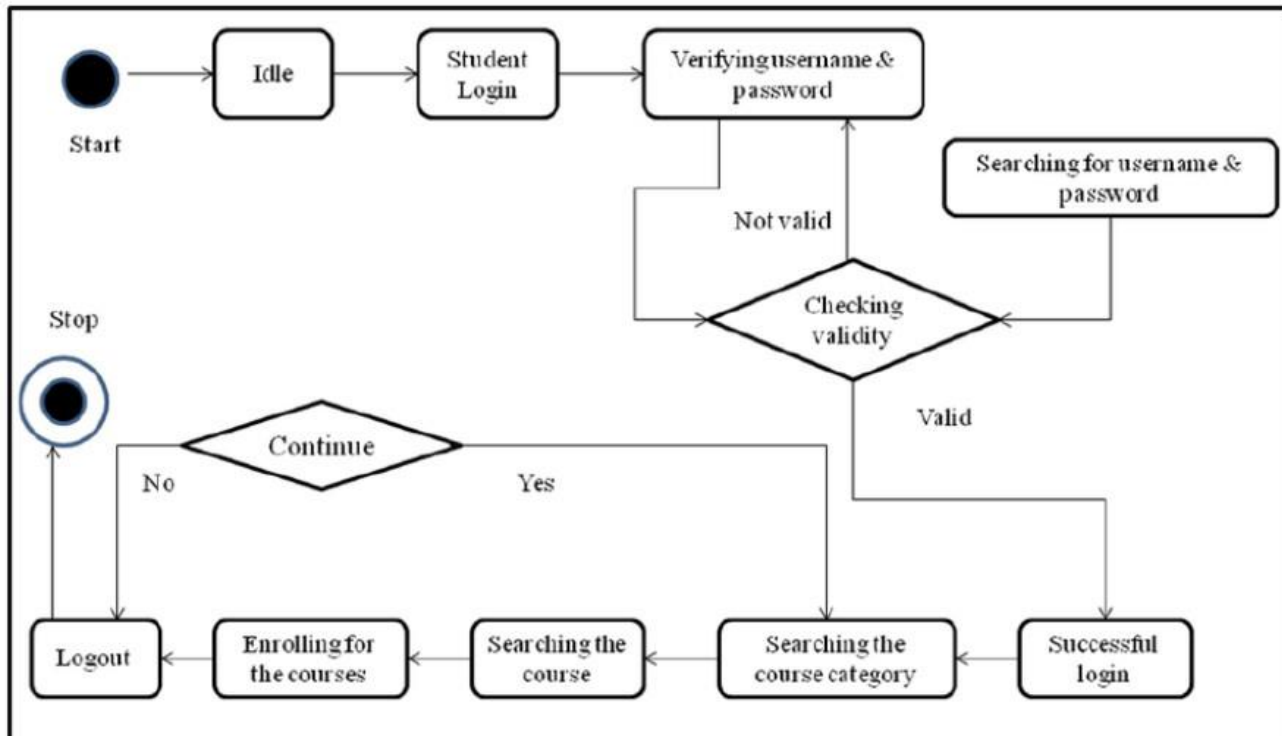
Result and Discussion:

In software engineering, the dynamic view of a system refers to how the system behaves and changes over time during its execution. It focuses on the runtime aspects of the software, such as the flow of control, interactions between different components, and the sequence of events that occur as the system executes.

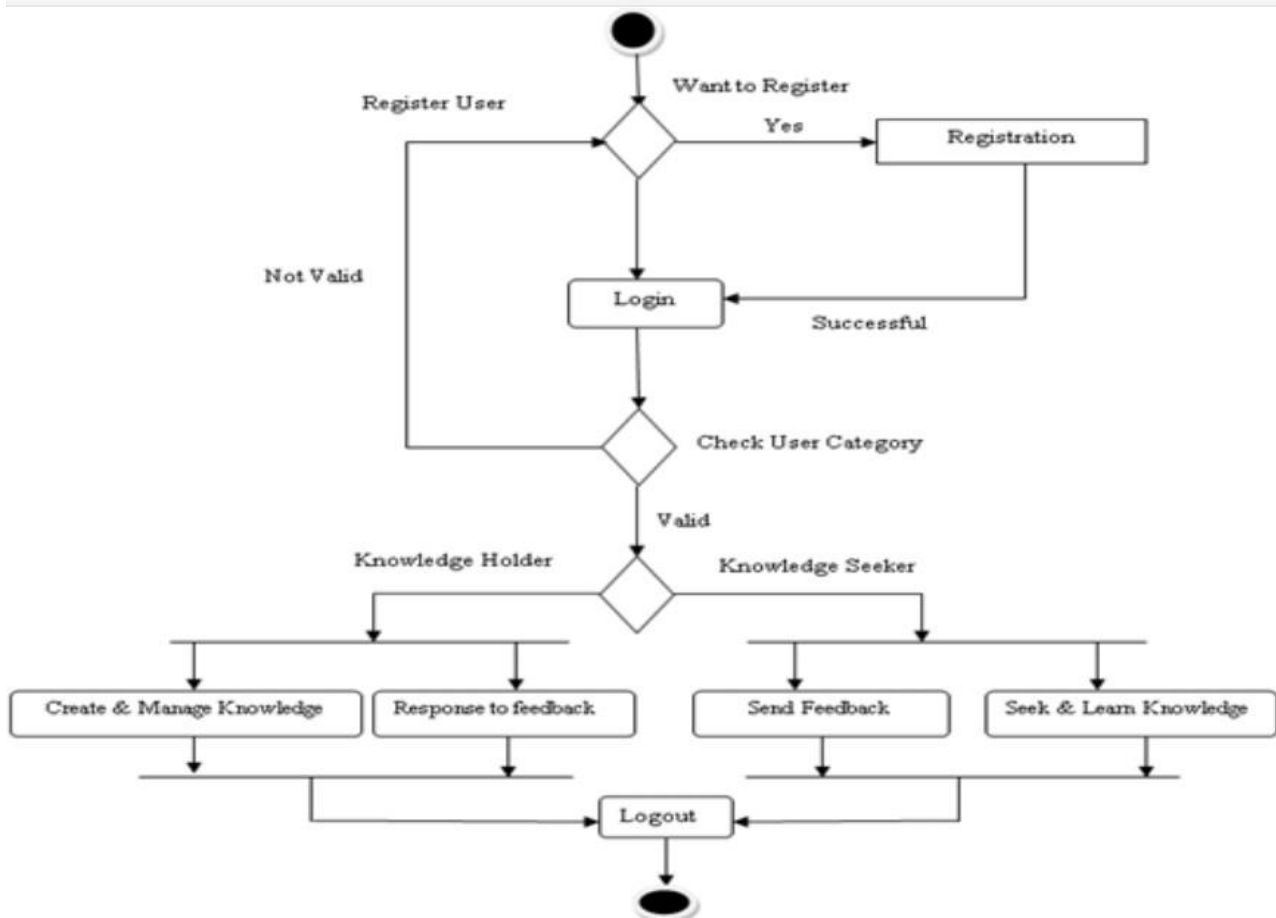
The dynamic view helps software engineers understand how the system responds to various inputs, how it handles different scenarios, and how it transitions between different states during its operation. This view is typically represented using diagrams like sequence diagrams, activity diagrams, state diagrams, and collaboration diagrams, which illustrate the dynamic behavior of the system.

By analyzing the dynamic view of a system, software engineers can identify potential issues such as concurrency problems, performance bottlenecks, and unexpected behavior. They can also use this understanding to optimize the system's performance, improve its reliability, and ensure that it meets the desired functional and non-functional requirements.

State Diagram :



Activity Diagram :



- LO 1: Identify the importance of state diagram. LO
 2: Draw activity diagrams for a given scenario.

Course Outcomes: Upon completion of the course students will be able to understand and demonstrate state and activity diagrams.

Conclusion:

The dynamic view of a system in software engineering focuses on its runtime behavior and changes during execution, aiding in understanding interactions, flow of control, and state transitions. It helps identify issues and optimize performance for meeting requirements.

For Faculty Use:

Correction Parameters	Formative Assessment [40%]	Timely completion of Practical [40%]	Attendance / Learning Attitude [20%]	
Marks Obtained				