Discussion-8 Compare a state machine diagram with a sequence diagram

Discussion Topic:

Compare a state machine diagram with a sequence diagram and discuss the differences between the two. Please include at least one diagram in your post.

My Post:

Hello Class,

In software engineering, Unified Modeling Language (UML) state machine diagrams, also known as state diagrams or statecharts, are used to model the dynamic of systems both in the problem space (MOPS) and the solution space (MOSS) (Unhelkar, 2018). UML State Machine Diagrams are helpful for visualizing how objects change state over time. This post explores UML State Machine Diagrams, including their notation, construction steps, common errors, and relationship to UML sequence diagrams.

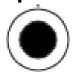
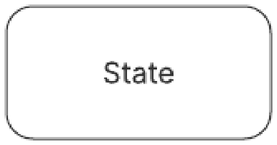
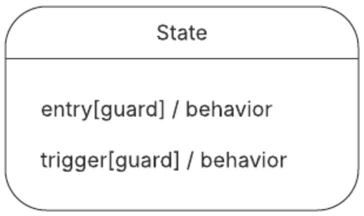
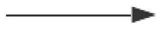
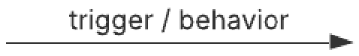
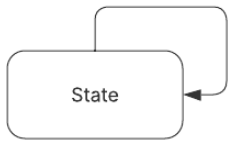
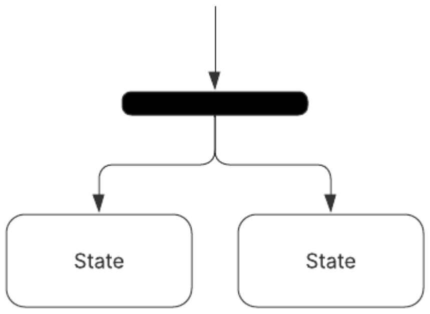
State Machine Diagrams Overview

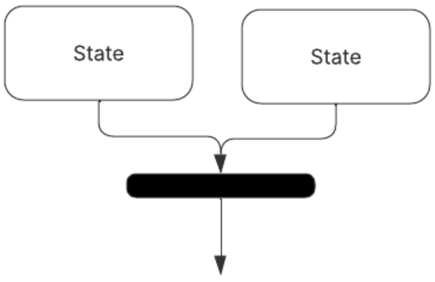
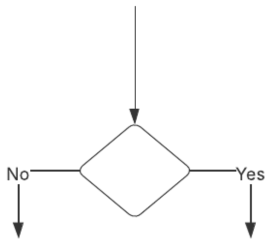
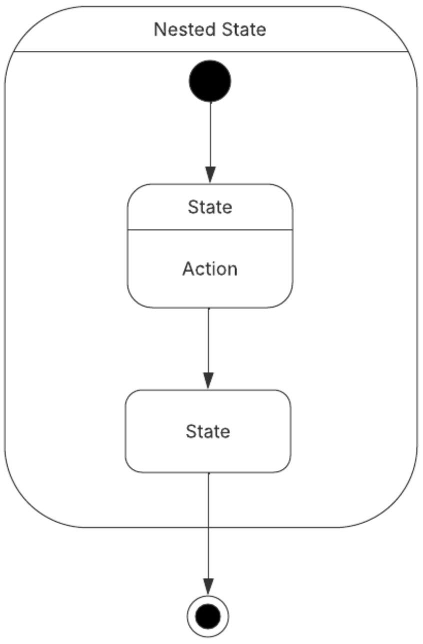
UML State Machine Diagrams (SMDs) are behavior diagrams that show the discrete behavior of a system by depicting the state transitions of a part of that system (UML®, n.d.). Two kinds of state machines are defined in UML. 2.5, behavioral state machine and protocol state machine. A behavioral state machine focuses on event-driven transition of an object (Dwivedi, 2019). A protocol state machine is a behavioral state diagram that specializes in modeling the sequence of event-driven transition of an object. This post focuses on behavioral state diagrams as they are more widely used and the notations of the two types are very similar.

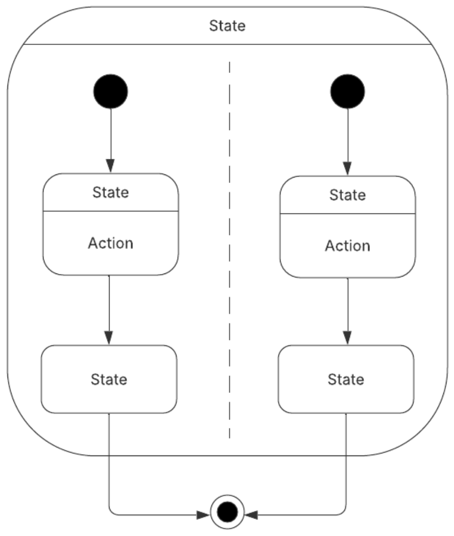
In software modeling, SMDs provide a dynamic illustration of the object within a system. They are used in MOPS (Model of Problem Space) to model the state states, conditions, and transitions of system entities, and in MOSS (Model of Solution Space), they are used to model the states, conditions, and transitions of objects including interface and control objects (Unhelkar, 2018). In other words, they are used to visualize the different states of objects within a system and how, in response to events, they transition between those states. Various components are utilized to create SMDs, the table below lists the main components that can be found in SMDs.

Table 1
UML SMD Basic Notation

Notation	Description
Start State ●	Represents a condition or situation in which an object starts existing.

Stop State 	End of the object's lifecycle
Boundary 	Represent state representing the condition of an object.
Composite State  State entry[guard] / behavior trigger[guard] / behavior	A state with internal activities. - entry: label how this state is entered - trigger: an event that initiates a transition from one state to another. - [guard]: a Boolean condition that allows a behavior to be performed. - Behavior: an activity
Transition 	The change from one state to another. Label with an event.
Event 	An event is a type of occurrence that based on a condition triggers a behavior that causes a transition (change of state). It can be external, internal to the system, or the passage of time.
Self-Transition 	A self-transition starts and ends in the same state, triggering exit and entry behavior within the same state.
Fork 	Represents an object transitioning two or more concurrent states.

Join 	Represent an object transitioning out from two or more concurrent states.
Decision Point 	Represents a point where a single transition splits into multiple transitions, based on guard conditions.
Composite State 	A composite state, also called a nested states, represents a state with sub-states within. This allows developers to represent hierarchical state structures.

Composite State with Concurrency  State State Action State Action State State	Represents a composite state with two or more sub-states concurrent sub-states within. This allows developers to represent concurrent hierarchical state structures.

Note: This table provides an illustration and a description of the basic notation elements that can be found within state machine diagrams. From several sources (Unhelkar, 2018; Dwivedi, 2019; GeeksForGeeks, 2025; Visual-Paradigm, n.d.; IBM, 2023, FastBitLab, 2022).

When building an SMD, it is generally a good practice to follow these steps:

1. Select an important/complex object.
2. Understand the different stereotypes present in the select object (Entity, boundary, control, or table).
3. Identify states based on attribute values from the class documentation.
4. Figure out how the object changes state.
5. Draw transitions between states.
6. Specify conditions for transitions, aka guards and related events and triggers.
7. Group related states hierarchically using composite states.
8. Clarify the diagram by adding notes.
9. Consider modeling other objects/use cases.

(Unhelkar, 2018)

The table below lists the most common errors that can be made while building a SMD and how to rectify them.

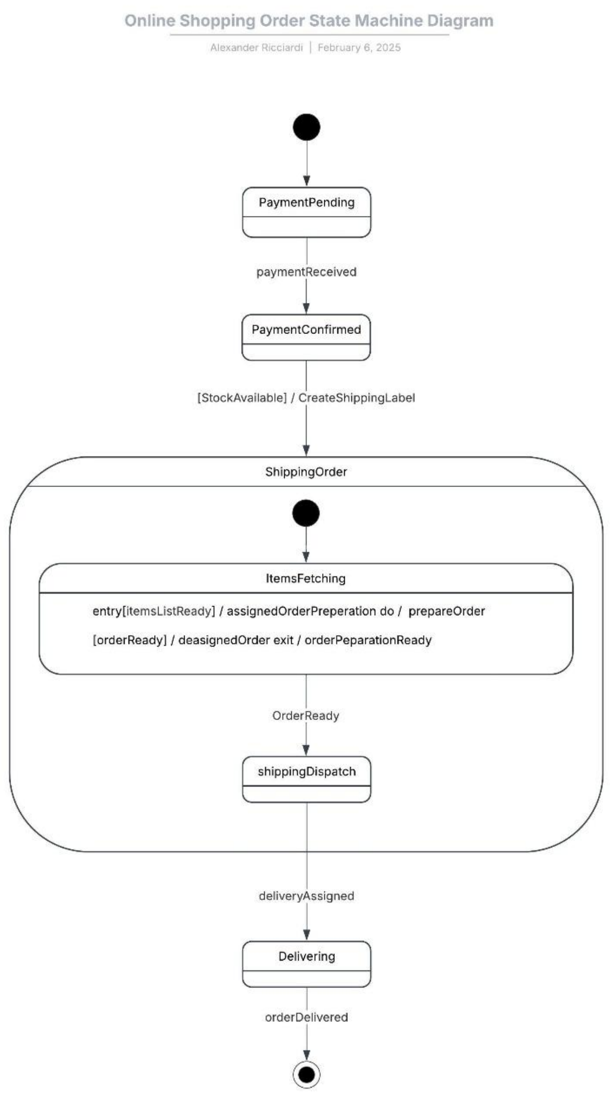
Table 2
SMD Common Error and Rectifications

Common Errors	Rectifying the Errors
Drawing an SMD like a flow chart (activity diagram)	List clearly all the states for ONE object. Ensure these states are not activities.
Having more than one start state	This is the single starting point to read the SMD. Remove all others (except within a nested substate).
Differentiating between self-transition and no transition	When a nested message hits an object but the object does not change state, it is a self-transition.
Not understanding an unconditional transition	Understand that it occurs by default when an activity is completed.
Confusing decision point (diamond) and a guard condition	Use a decision point when a transition can occur to potentially more than one state.
Showing states for a class rather than an object	Although the class appears in discussions and in the modeling tool, the states are always for an object.
Not showing a start state within a nested state	It is important to treat a nested substate diagram similar to a state diagram; therefore, it will have a start state.

Note: The table lists the most common errors made while constructing an SMD and how to rectify them. From “Chapter 12 — Dynamic modeling With state Machine Diagrams. Software Engineering with UML” by Unhelkar (2028). Modified.

The diagram below is an example of an SMD illustrating an online shopping order’s states.

Figure 1
SMD Example of an Online Shopping Order


Online Shopping Order State Machine Diagram
Alexander Ricciardi | February 6, 2025
PaymentPending
paymentReceived
PaymentConfirmed
[StockAvailable] / CreateShippingLabel
ShippingOrder
ItemsFetching
entry[itemsListReady] / assignedOrderPreperation do / prepareOrder
[orderReady] / deasignedOrder exit / orderPeparationReady
OrderReady
shippingDispatch
deliveryAssigned
Delivering
orderDelivered

Note: The figure is an illustration of a UML State Machine Diagram (SMD) depicting the states of an online shopping order.

SMDs vs Sequence Diagrams

Another type of UML diagram used to represent the dynamic aspect of a system is the sequence diagram. Sequence diagrams "provide a dynamic illustration of object interactions within a system. In SE, they are usually based on UML class diagrams and are used primarily to show the interactions between objects (classes) in the chronological order in which those interactions occur" (Ricciardi, 2025, p.1). These diagrams are also used in the MOPS to model the behavior of systems from the actors' (users) perspective and in the MOSS to model objects' interactions in detail; details such as the sequence of messages (methods) within a system, the parameter list within messages, and the return values of the messages. Both the SMDs and sequence diagrams are useful for modeling the dynamic aspects of a system and depending on the specific needs of analysis and design task one is more suited than the other one. For example, to model the internal behavior of a single object, a state machine diagram is best, whereas to model multiple objects and the sequence of operations within a system, a sequence diagram is the better choice.

To summarize, SMDs are used to understand and model the dynamic behavior of systems, particularly the state changes/transitions of individual objects. They help to model the states, transitions, events, and guard conditions of a system. Thus, software engineers need to understand the common errors that can occur when building SMDs and best practices, as well as the relationship to other UML diagrams like sequence diagrams. Ultimately, SMDs are a powerful tool for analyzing (in MOPS) and designing (n the MOSS) the dynamic behavior of individual objects within systems.

-Alex

References:

Dwivedi, N. (2019, September 9.). Software design: Modeling with UML
State machine diagram. *Modeling with the Unified Modeling Language (UML).* LinkedIn Learning. https://www.linkedin.com/learning/software-design-modeling-with-uml/state-machine-diagram?u=2245842

FastBitLab (2022, January 20). *FSM Lecture 11- UML state machine types of transitions.* FasBitLab. https://fastbitlab.com/fsm-lecture-11-uml-state-machine-types-of-transitions/

GeeksforGeeks (2025, January 3). *State machine diagrams | Unified Modeling Language (UML).* GeeksforGeeks. https://www.geeksforgeeks.org/unified-modeling-language-uml-state-diagrams/

IBM (2023, September 2018). Creating transitions between states. *IBM DevOps Model Architect*. IBM Documentation. https://www.ibm.com/docs/en/dma?topic=diagrams-creating-transitions-between-states

Ricciardi, A. (2025, January 27). *A Guide to UML Sequence Diagrams: Notation, strengths, and Limitations*. Code Chronicles - Omegapy. https://www.alexomegapy.com/post/a-guide-to-uml-sequence-diagrams-notation-strengths-and-limitations

UML® (n.d.) State machine diagrams. UML diagrams. UML Diagrams Org. https://www.uml-diagrams.org/state-machine-diagrams.html

Unhelkar, B. (2018). Chapter 12 — Dynamic modeling with state machine diagrams. *Software engineering with UML.* CRC Press. ISBN 9781138297432

Visual-Paradigm (n.d.). *What is state machine diagram?* Visual-Paradigm . https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/