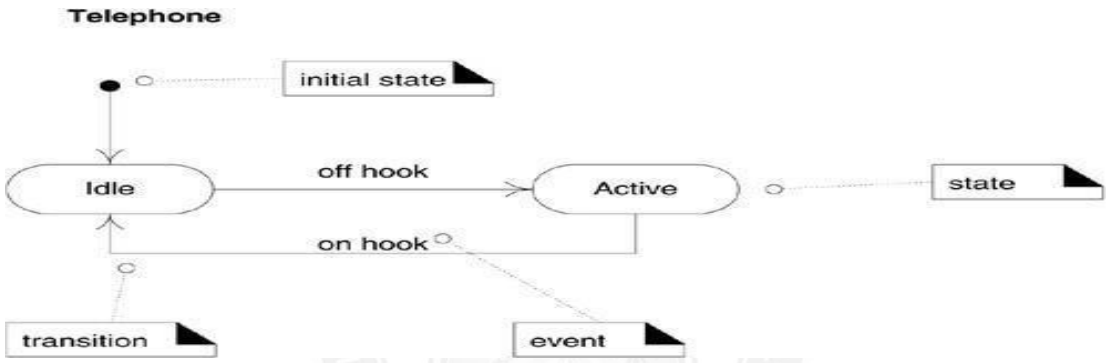
UML STATE MACHINE DIAGRAMS AND MODELLING

A UML state machine diagram illustrates the interesting events and states of an object, and the behavior of an object in reaction to an event. Transitions are shown as arrows, labeled with their event. States are shown in rounded rectangles. It is common to include an initial pseudo-state, which automatically transitions to another state when the instance is created.



State machine diagram for a telephone

A state machine diagram shows the lifecycle of an object: what events it experiences, its transitions, and the states it is in between these events. Therefore, we can create a state machine diagram that describes the lifecycle of an object at arbitrarily simple or complex levels of detail, depending on our needs

Definitions: Events, States, and Transitions

An event is a significant or noteworthy occurrence. For example:

- A telephone receiver is taken off the hook.

A state is the condition of an object at a moment in time the time between events. For example:

- A telephone is in the state of being "idle" after the receiver is placed on the hook and until it is taken off the hook.

A transition is a relationship between two states that indicates that when an event occurs, the object moves from the prior state to the subsequent state. For example:

- When the event "off hook" occurs, transition the telephone from the "idle" to "active" state.

Guidelines : To Apply State Machine Diagrams:

Object can be classified into

1) **State-Independent Object** - If an object always responds the same way to an event, then it is considered state-independent (or modeless) with respect to that event. The object is state-independent with respect to that message.
2) State-Dependent Objects - State-dependent objects react differently to events depending on their state or mode.

   Guideline : Consider state machines for state-dependent objects with complex behavior, not for state-independent objects . For example, a telephone is very state-dependent. The phone's reaction to pushing a particular button (generating an event) depends on the current mode of the phoneoff hook, engaged, in a configuration subsystem, and so forth.

Modeling State-Dependent Objects : state machines are applied in two ways:

1. To model the behavior of a **complex reactive object** in response to events.
2. To model **legal sequences of operations** protocol or language specifications.
   - This approach may be considered a specialization of #1, if the "object" is a language, protocol, or process. A formal grammar for a context-free language is a kind of state machine.

1. Complex Reactive Objects

a) Physical Devices controlled by software
- Phone, car, microwave oven: They have complex and rich reactions to events, and the reaction depends upon their current mode.

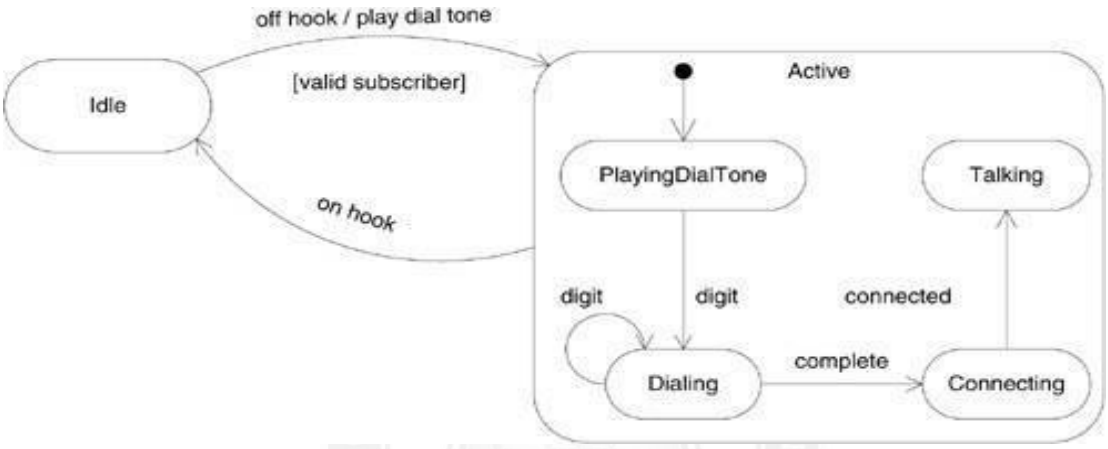
b) Transactions and related Business Objects
- How does a business object (a sale, order, payment) react to an event? For example, what should happen to an Order if a cancel event occurs? And understanding all the events and states that a Package can go through in the shipping business can help with design, validation, and process improvement.

c) Role Mutators : These are objects that change their role.

A Person changing roles from being a civilian to a veteran. Each role is represented by a state.

Example 1: Physical Devices / Nested States – Telephone Object

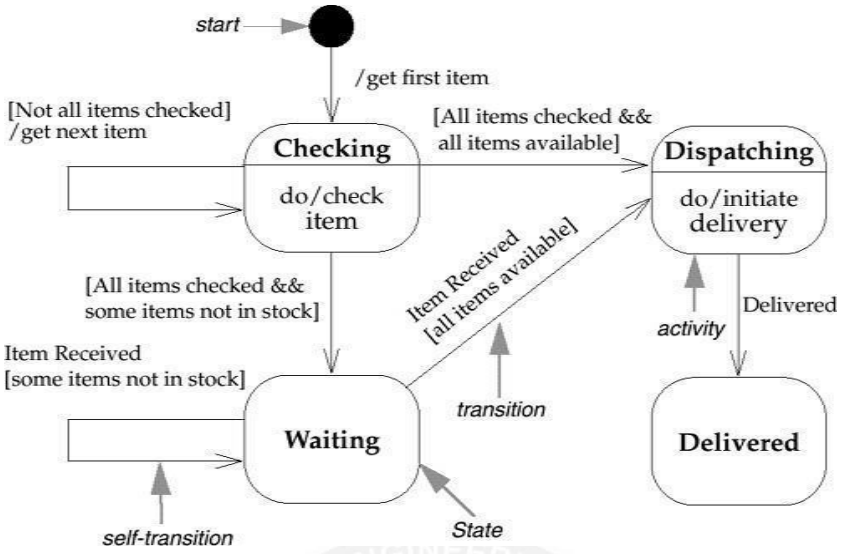
A state allows nesting to contain substates; a substate inherits the transitions of its superstate (the enclosing state. It may be graphically shown by nesting them in a superstate box..



For example, when a transition to the Active state occurs, creation and transition into the PlayingDialTone substate occurs. No matter what substate the object is in, if the on hook event related to the Active superstate occurs, a transition to the Idle state occurs.

Example 2: Transactions and related Business Objects

Order Processing System – Order Object

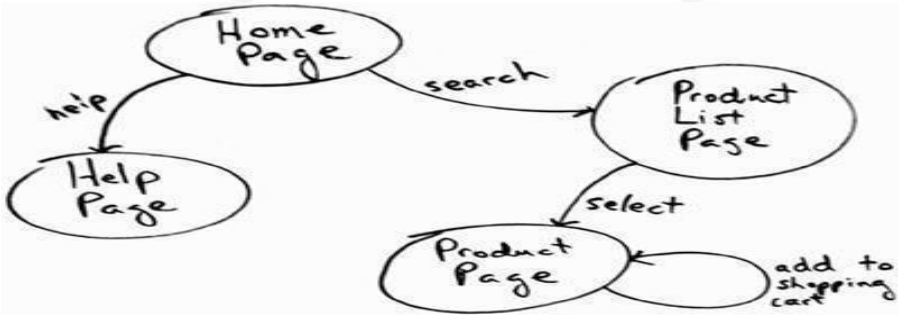


Protocols and Legal Sequences

a) **Communication Protocols**
   - TCP, and new protocols, can be easily and clearly understood with a state machine diagram. For example, a TCP "close" request should be ignored if the protocol handler is already in the "closed" state.

b) **UI Page/Window Flow or Navigation** When doing UI modeling, it can be useful to understand the legal sequence between Web pages or windows;

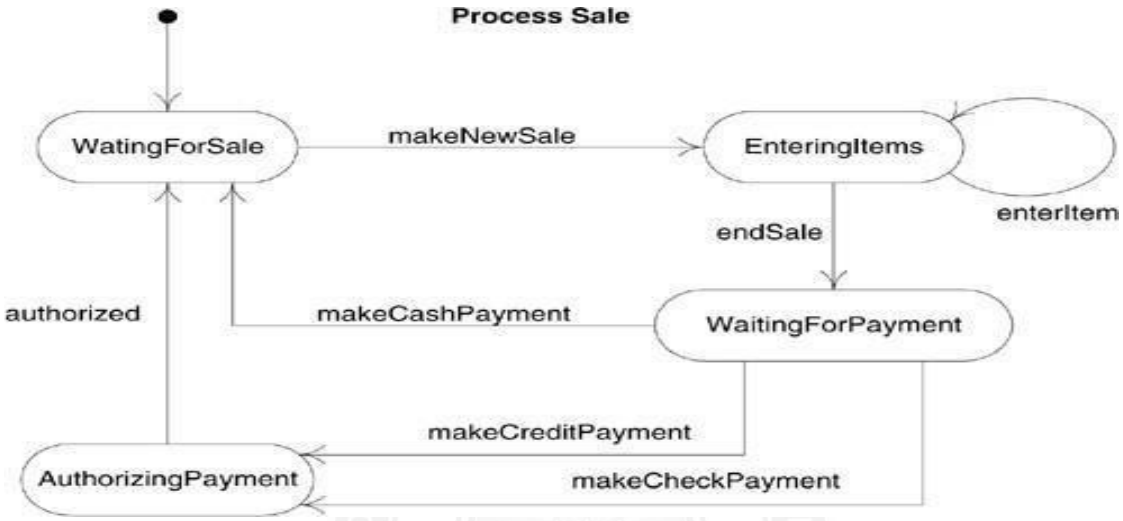
Applying a state machine to Web page navigation modeling.



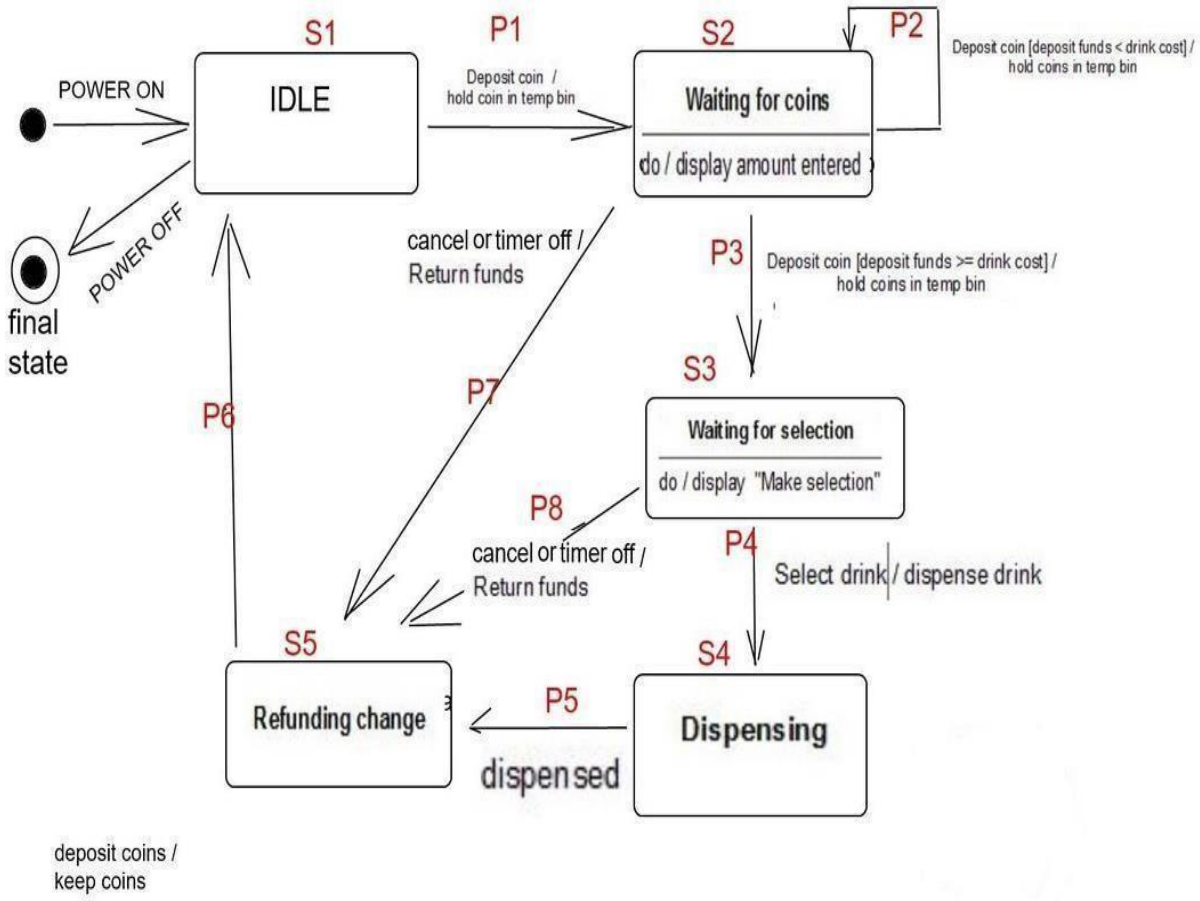
UI Flow Controllers or Sessions These are usually server-side objects representing an ongoing session or conversations with a client. For example, a Web application that remembers the state of the session with a Web client and controls the transitions to new Web pages, or the modified display of the

current Web page, based upon the state of the session and the next operation that is received.

d) **Use Case System Operations** Do you recall the system operations for Process Sale: makeNewSale, enterItem etc. These should arrive in a legal order; for example, endSale should only come after one or more enterItem operations.



STATE DIAGRAM : COFFEE VENDING MACHINE



WHEN TO USE STATE DIAGRAM

State chart diagram is one of the five UML diagrams used to model the dynamic nature of a system. They define different states of an object during its lifetime and these states are changed by events. State chart diagrams are useful to model the reactive systems. Reactive systems can be defined as a system that responds to external or internal events.

State chart diagram describes the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered. The most important purpose of State chart diagram is to model lifetime of an object from creation to termination.

State chart diagrams are also used for forward and reverse engineering of a system. However, the main purpose is to model the reactive system.

The main usage can be described as −

- To model the object states of a system.
- To model the reactive system. Reactive system consists of reactive objects.
- To identify the events responsible for state changes.
- Forward and reverse engineering.

Example: