State Modelling and Advanced state Modelling

Unit 2

Introduction: State Modelling

- State model describes the sequence of operations that occur in response to external stimuli.
- The state model consists of multiple state diagrams, one for each class with temporal behaviour.
- It express the state diagram for graphical representation for state machine that relates **events** and **states**.
  - Events represents external Stimuli
  - State represents values of objects.

EVENTS

- An Event is a occurrence at a point in time.
- By definition, an event happens instantaneously with regard to the time scale of an application.
  - One event may logically precede or follow another(related) or
  - two events may be unrelated(concurrent).

- For example,
- The User clicks the right button of mouse
- Train departs from Mysore to Bangalore.

CONTD..

- Events include error conditions as well as normal occurrences.

- For example,

  motor jammed,

  transaction aborted etc..

Types of Events

- There are three types of events are
  1. Signal event
  2. Change event
  3. Time event

1. Signal Event

- A signal is an explicit one-way transmission of information from one object to another.

- Signals are by definition, different from a subroutine or method which returns a value.
  - While a subroutine may be initiated by receipt of a signal, the return value is, itself, a separate message.

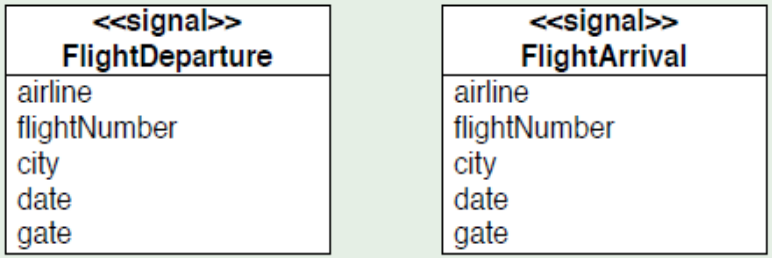
  - The **signal will not expect a reply to confirm receipt**; such formalisms must be extensions of the designed software.

CONTD..

- A signal event is the event of **sending or receiving a signal.**
- signal is a message; “A signal event is an occurrence of a signal in time".
- Every signal event is a unique occurrence, but we can group them into signal classes. We use the stereotype of signal in guillemets (<< signal>>)

Example

FlightDeparture: airline, flightNumber, city, date, gate
FlightArrival: airline, flightNumber, city, date, gate



2. Change Event

- A change event is an event that is caused by the satisfaction of a Boolean expression.
- The intent of a change event is that the condition is continuously tested, and that whenever the expression changes from false to true, the event happens.
- UML notation for a change event is the keyword "**when**" followed by a parenthesized boolean expression.

Example

Sample change events

- when(roomTemperature > heatingSetPoint)
- when(roomTemperature < heatingSetPoint)
- when(batteryPower < lowerLimit)
- when(tirePressure < minimumPressure)

3. Time Event

- A time event is an event caused by the occurrence of an absolute time or the elapse of a time interval.

Example

Time event examples

- when(date = "23:58:14 January 1, 2000")
- after(1.04 seconds)

UML notation:

- for an absolute time, is the keyword "**when**" followed by parenthesized expression involving time.
- For a time interval, it is the keyword "**after**" followed by a parenthesized expression that evaluates to a time duration.

States

- A state is an abstraction of the values and links of an object.
- States generally refer to a verb with a suffix "ing" (e.g., boiling, waiting, dialling etc.) or duration of some condition (eg. Powered, BelowFreezing),
- whereas events generally refer to an action (e.g., StoveTurnedOn). Note that events may be generated by state changes (e.g., change events).
- The UML notation for a state is a rounded box containing an (optional) state name.

Event Vs State

Events represent points in `time‘; states represent interval of time, ie., the abstract state of an object (or collection of objects). Consider the below diagram.

Example (Some Sample States)

Example states:

- Powered
- Boiling
- InFlight
- OnApproach
- Boarding

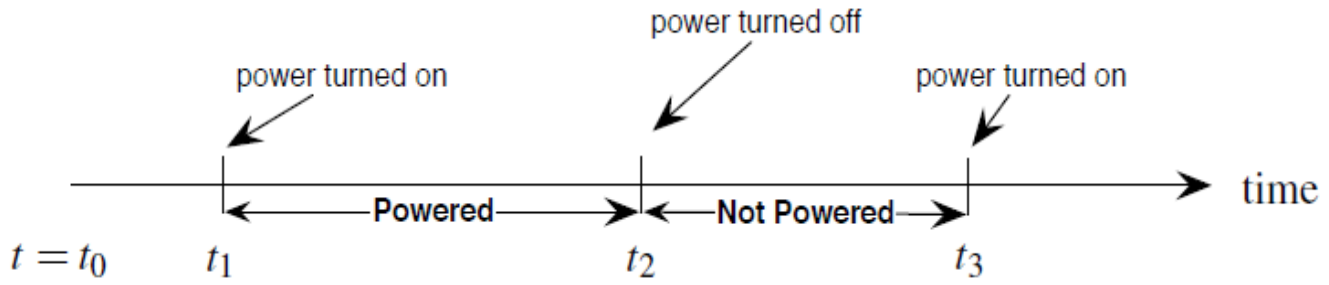


Figure: Power up, power down

- The objects in a class have finite number of possible states. Each object can only be in one state at time.
- Object pass through one or more states during the life time.
- A state specifies the response of an object to input events.
- An events are ignored in a state, expect those for which behavior is explicitly prescribed.
- Eg: if a digit is dialed in state Dial tone, the phone line drops the Dial tone & enters state **dialing**

Events Vs States

- Events represent points in time
  - State represent intervals of time.
- State corresponds to the **interval between two events** received by an object.

- Both events & states depend on the level of abstraction.
- Eg: A travel agent planning an itinerary (plan of journey) would treat each segment of a journey as a Single event.

Transitions & Conditions

- A transition is an instantaneous change from one state to another.
- UML notation:

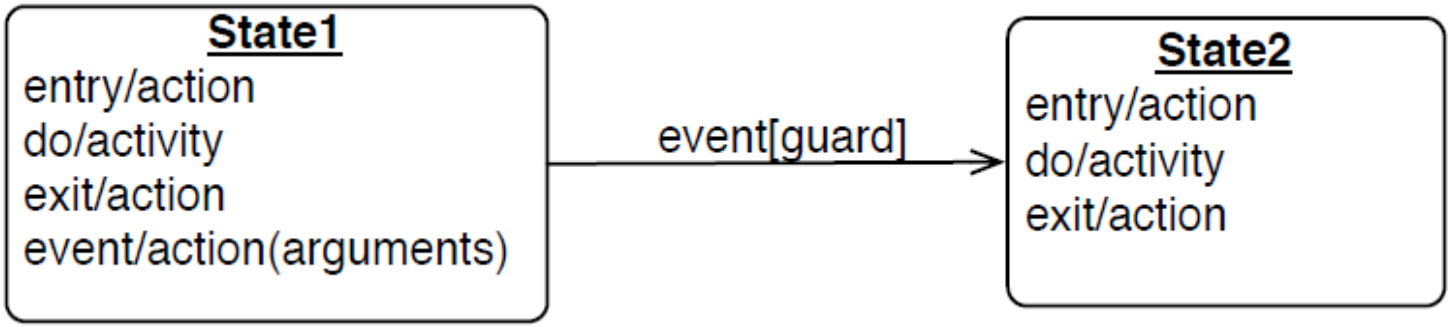
  drawn as a line from the origin state to the target state.
- A transition is said to fire upon the change from the **source state** to the **target state**.
  - Ex: when a called phone is answered, the phone line transitions from the Ringing state to Connected State.

- Generally, the origin and target states are different, but not necessarily. The choice of next state depends both on the original state, and the event received.
- An event may cause multiple objects to transition; in this case, you can consider that they transition concurrently.

CONTD…

A **guard condition** is a Boolean expression that must be true in order for a transition to occur.

The syntax for guard and event along a transition is event[guard].



Example

- Another example :
- When you go out in the morning, if the temperature is below freezing, then put on your gloves.

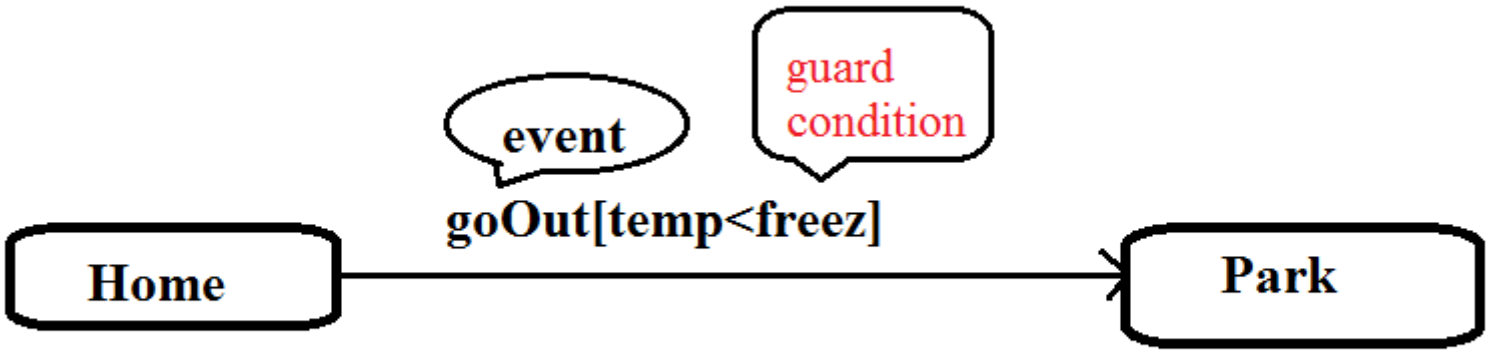
Example

- Another example :

- When you go out in the morning (Event), if the temperature is below freezing (guard condition), then put on your gloves(state).

UML notation for a transition:

- Use a line from origin state to the target state, arrowhead points to the target state.
- An event may label the transition and be followed by an optional guard condition in square brackets.



State Diagrams

- A state diagram is a graph whose nodes are states and whose directed, annotated arcs are transitions between states.

Some constraints behind the structure of a state diagram:

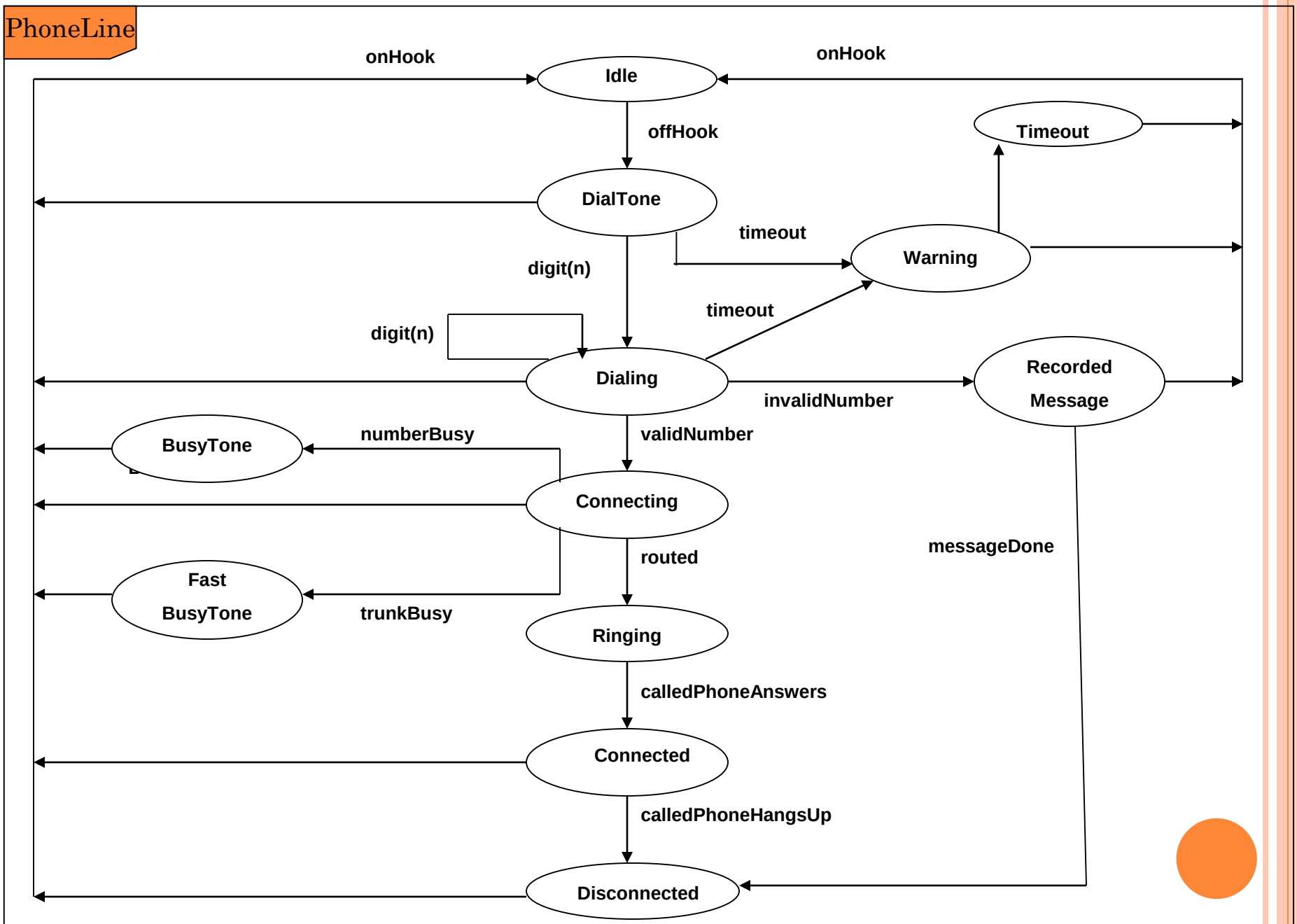
- State names must be unique within the scope of a state diagram
- All objects in a class execute the state diagram for that class, which models their common behaviour

CONTD..

- The state model consists of multiple state diagrams, one state diagram for each class.
- State diagram can represent **continuous loop** or **one-shot life cycle**.

Other Constraints:

- State diagrams must match on their interfaces events and guard conditions
- State diagrams interact by passing events and through side effects of guard conditions



State diagram for a telephone line

One-shot State Diagrams

- One shot diagrams represents objects with finite lives and have **initial** & **final** states.
- A one shot diagram has a start - depicted as a solid circle - corresponds to object creation
- A final state is depicted with a bulls eye - corresponds to object destruction.
- In two ways we can demonstrate one shot state diagram:
  - **with Finite lives**
  - **By using Entry and Exit points**

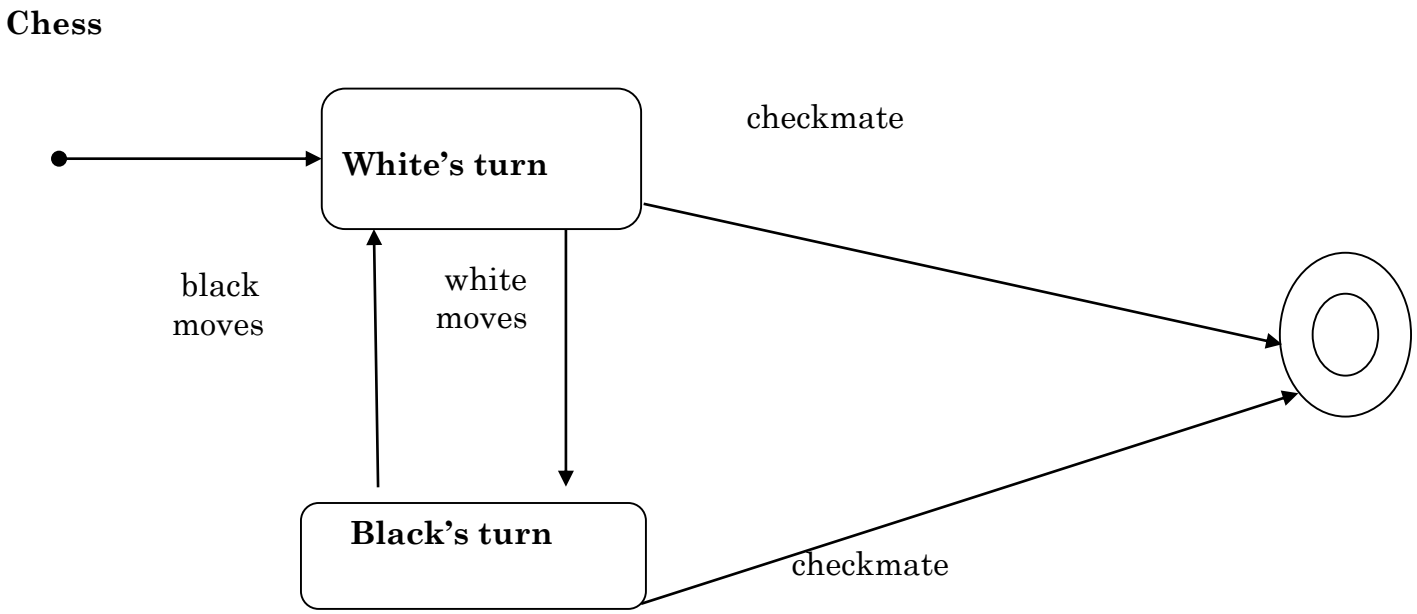
A) With Finite lives

- One-shot state diagrams represent objects with finite lives and have initial and final states.

- The initial state is entered on **creation** of an object; entry of the final state implies **destruction** of the object.

- The next figure shows a simplified life cycle of a chess game with a default initial state (solid circle) and a default final state (bull's eye)

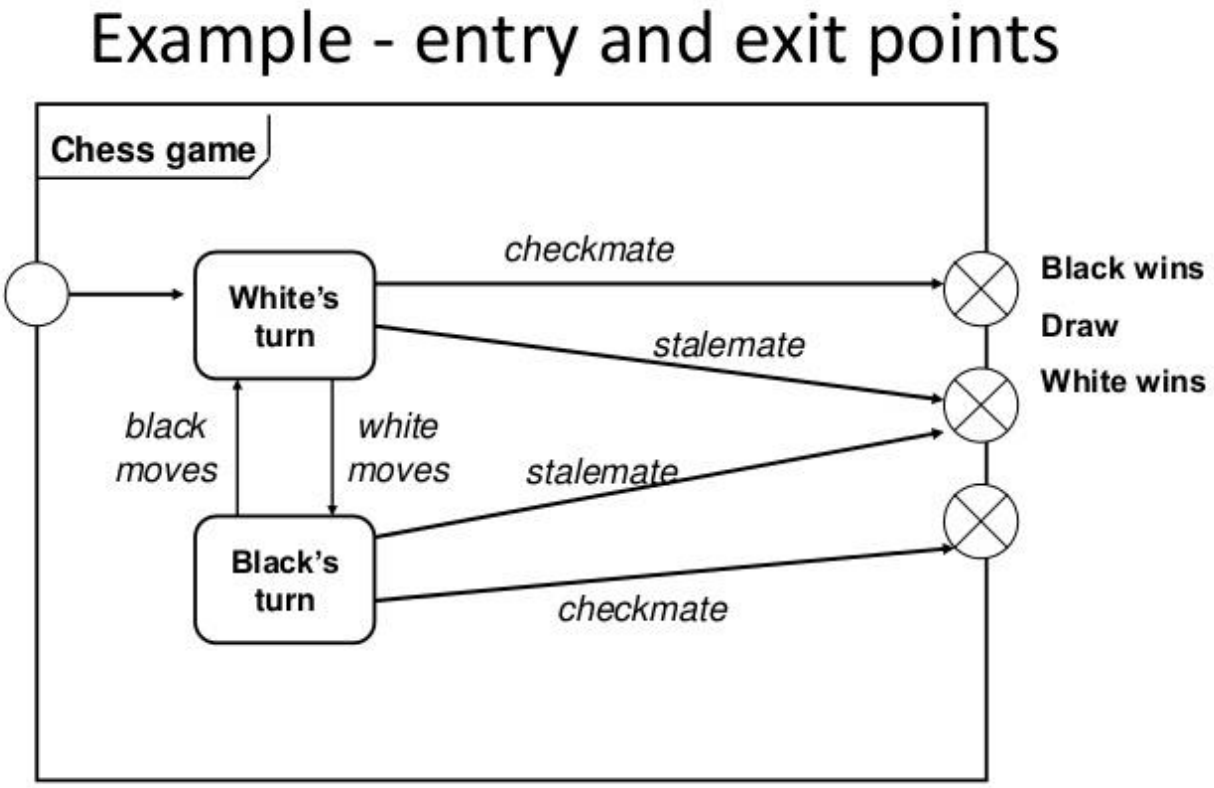
CONTD..


Chess
White's turn
checkmate
black
moves
white
moves
Black's turn
checkmate

B) Entry & Exit Points

- As an alternate notation, you can indicate **initial and final states** via entry and exit points.
- In the next figure, the *start* entry point leads to white's first turn, and the chess game eventually ends with one of three possible outcomes.
- Entry points (hollow circles) and exit points (circles enclosing an "x") appear on the state diagram's perimeter and may be named.

CONTD…


Example - entry and exit points
Chess game
White's turn
Black's turn
checkmate
stalemate
black moves
white moves
stalemate
checkmate
Black wins
Draw
White wins

State Diagram Behaviour

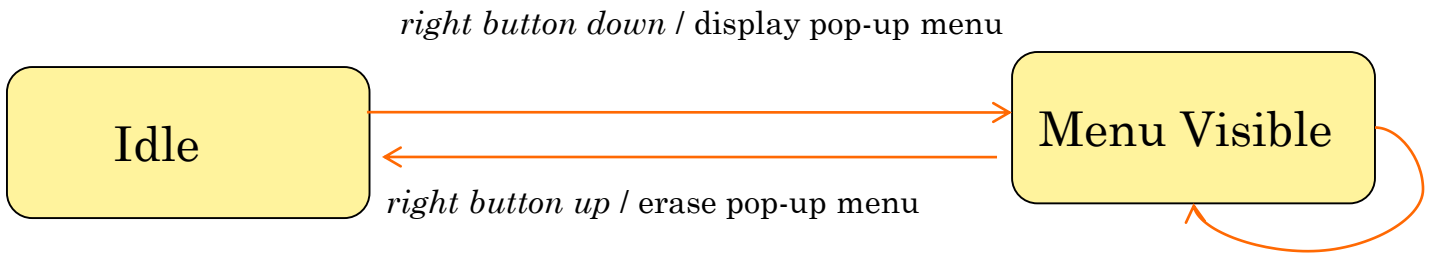
- Activity Effects
- Do-Activities
- Entry and Exit Activities
- Completion Transition
- Sending Signals
- Sample State Diagram with Activities

1. Activity effects:

- An effect is a **reference to a behavior** that is executed in response to an event.
- An activity is the actual behavior that can be invoked by any number of effects.
- For Ex: DisconnectPhoneLine might be an activity that is executed in response to an onHook event.

Activity Effects...

- The notation for an activity is a **slash ("/ ")** and the **name (or description)** of the activity, following the **event** that causes it.
- The keyword ***do*** is reserved for indicating an ongoing activity and may not be used as an event name



2. DO-ACTIVITIES

- A ***do-activity*** is an activity that continues for an extended time.
- A do-activity can only occur within a state and cannot be attached to a transition.
- For example, the **warning light** may flash during the **paper jam state** for a copy machine.

CONTD..

- do-activities include **continuous operations**, such as
  - displaying a picture on a television screen or monitor(eg. Screen saver).
  - as well as **sequential operations** that **terminate by themselves** after an interval of time, such as closing a valve.

- The notation "**do/**" denotes a do-activity that may be performed for all or part of the duration that an object is in a state.

CONTD..

- A do-activity may be interrupted by an event that is received during its execution; such an event may or may not cause a transition out of the state containing the do-activity.

Paper jam

do / flash warning light

3. **Entry and Exit Activities:**

- As an alternative to showing activities on transition, you can bind activities to entry or to exit from a state.
- For eg, fig shows **control of a garage door opener.**
  - The user generates depress events with a push button to open & close the door.
  - Each event reverse the direction of a door, but for safety the door must open fully before it can be closed. The control generates motorup and motor down activities for the motor.
  - The motor generates door open and door closed when motion has been completed.

3. Entry and Exit Activities

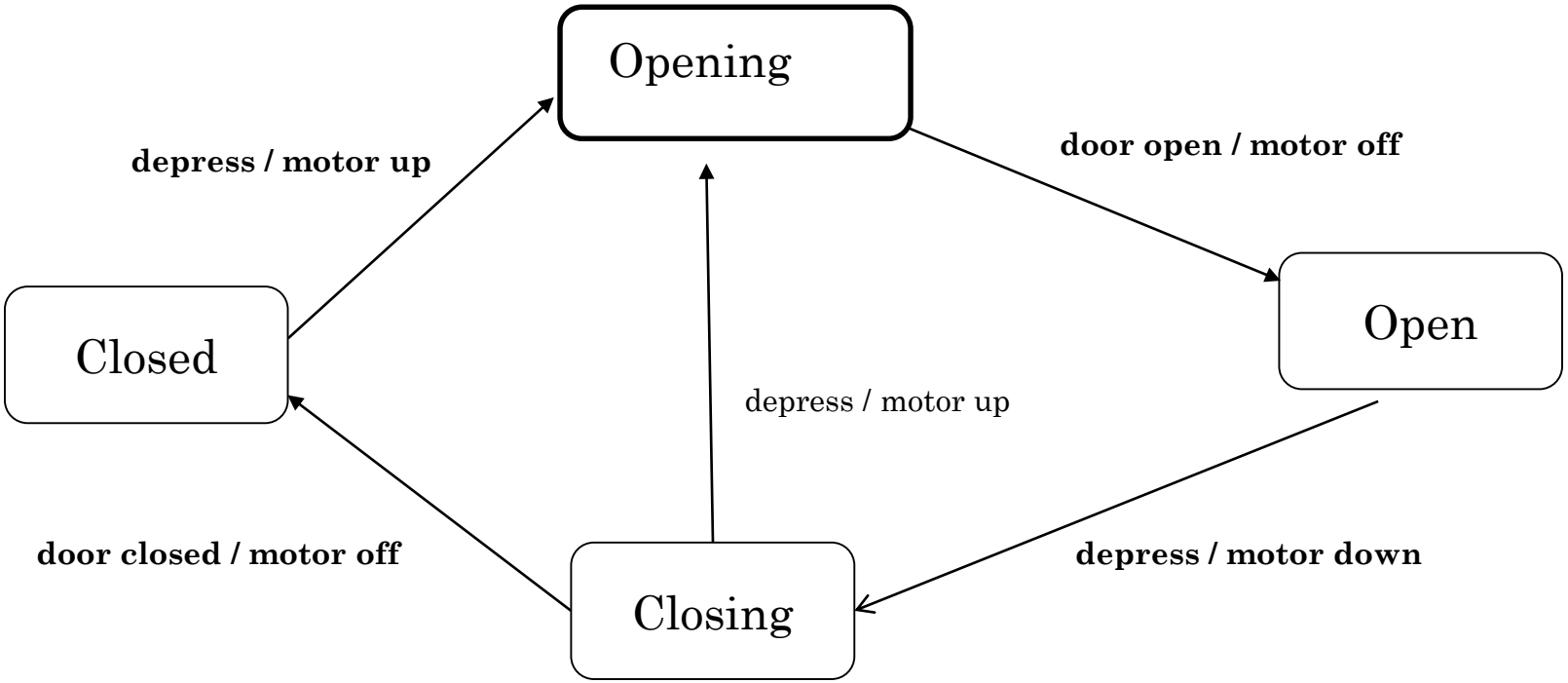
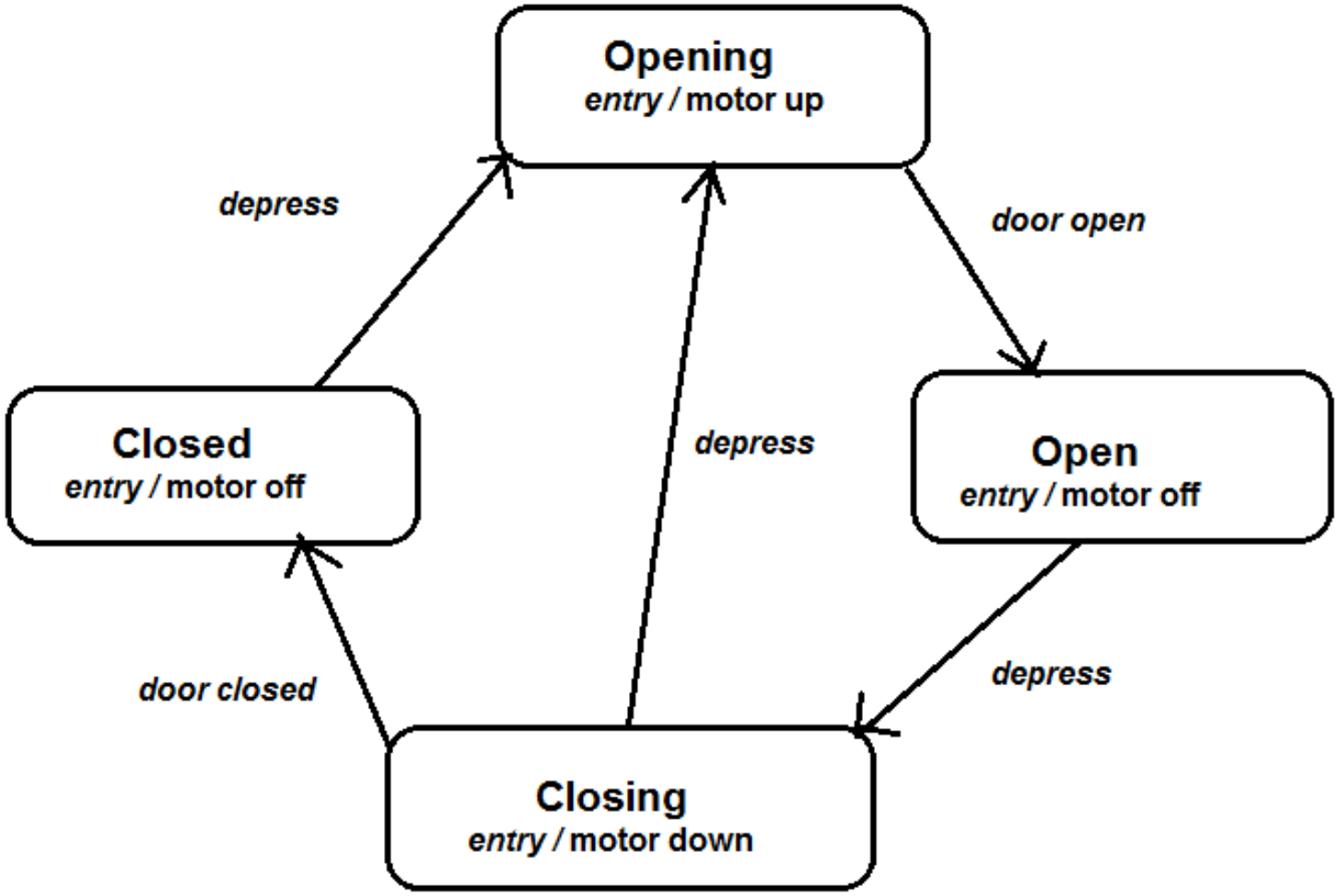
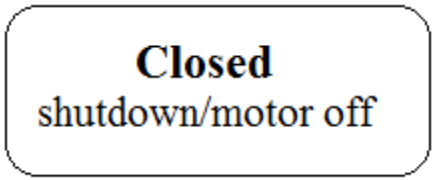


Fig: State diagram using entry and exit activities

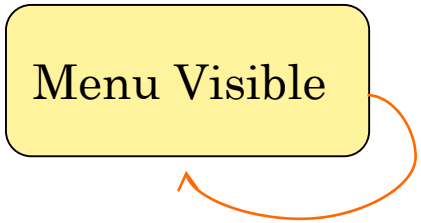


- Exit activities are less common than entry activities, but they are occasionally useful.
- Exit activity is shown inside the state box following the keyword exit & a " / " character.

- In general, there is a difference between **event within a state** and **self-transition**.



Event within a state



***cursor moved* / highlight menu item**

Self transition

4. Completion Transition

- The sole purpose of a state is to perform a sequential activity. When the activity is completed, a transition to another state fires.

- "An arrow without an event name indicates an automatic transition that fires when the activity associated with the source state is **completed**".

- Such unlabeled transition are called **Completion transitions**, because they are triggered by completion of activity in source state.

- Ex: Online exam with result.

CONTD..

- A guard condition is tested only once, when the event occurs.
- If a state has one or more completion transitions, but none of the guard conditions are satisfied, then the state remains active and may become "stuck"- the completion event does not occur a second time, therefore no completion transition will fire later to change the state.

CONTD…

- If a state has completion transition leaving it, normally the guard conditions should cover every possible outcome.
- The key word *else* can be used to apply if all the other conditions are false.
- Do not use a guard condition on a completion transition to model waiting for a change of value. Instead model the waiting as a change event.

5. Sending signals

- A system of objects interacts by exchanging signals
- The activity "send *target.S(attributes*)" sends signal *S* with the given attributes to the target object or objects.
- For example, the phone line sends a *connect(phonenumber)* signal to the switcher when a complete phone number has been dialed.

CONTD..

- A signal can be directed at a set of objects or a single object.

- If the target is a set of objects, each of them receives a separate copy of the signal concurrently, and each of them independently process the signal and determines whether to fire a transition.

CONTD..

- If the signal is always directed to the same object, the diagram can omit the target.
- For ex: the door may or may not remain open if the button is pressed at about the time the door becomes fully open
- If an object can receive signals from more than one object, the order in which concurrent signals are received may affect the final state; this is called a ***race condition***.

CONTD..

- Concurrent systems frequently contain unwanted race conditions that must be avoided by careful design.

- A requirement of two signals being received simultaneously is never a meaningful condition in the real world, as slight variations in transmission speed are inherent in any distributed system.

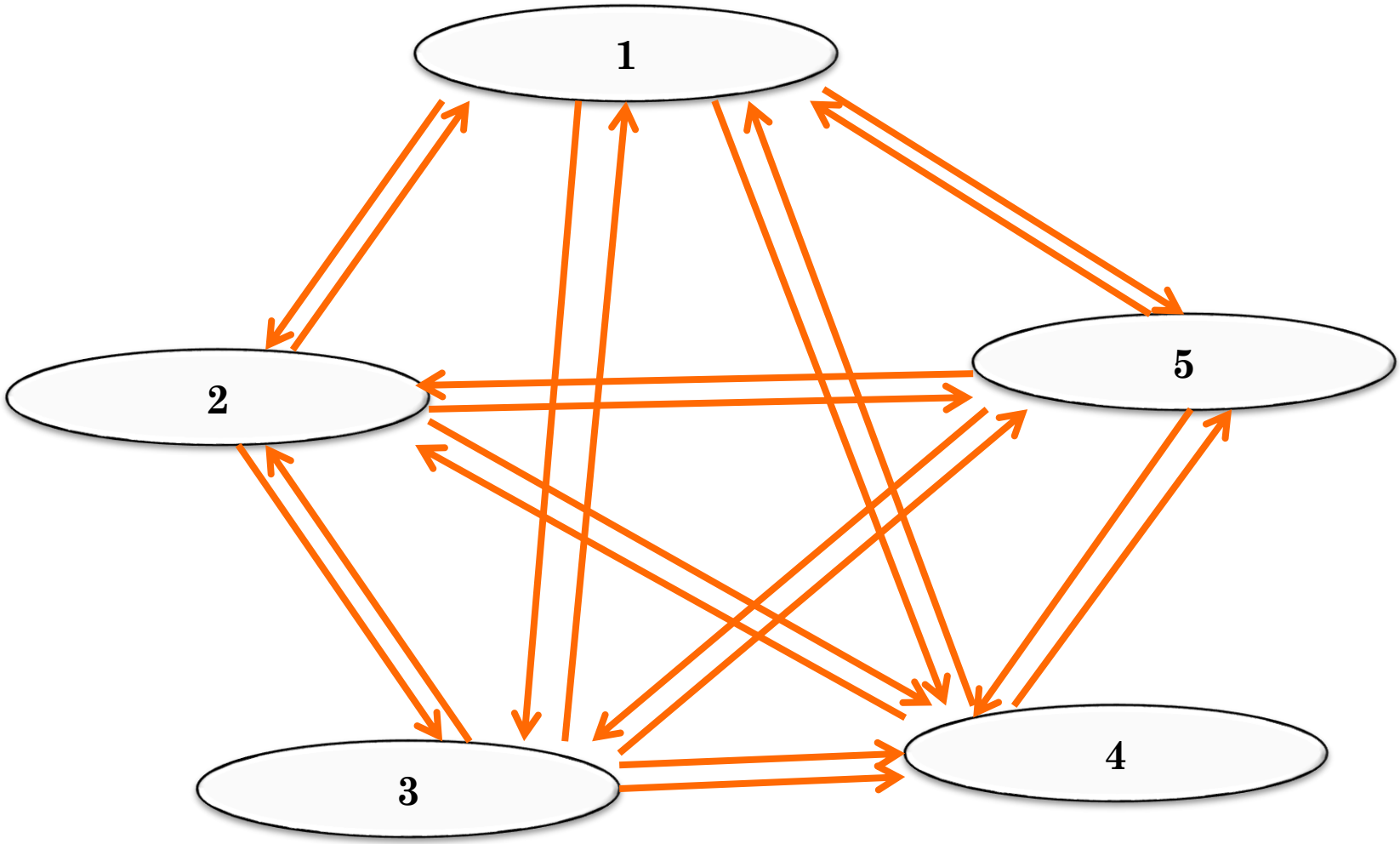
Advanced State Modeling

- **Contents:**
  - Problems with flat State Diagrams
  - Expanding States
  - Nested States
  - Signal Generalization
  - Concurrency

CONTD..

Problems with flat State Diagrams:

- Conventional state diagrams are sufficient for describing simple systems but need additional power to handle large problems.
  - Consider an object with ‘n’ independent **Boolean *attributes*** that affect control .
  - An object with single flat state diagram would require $\mathbf{2^n}$ states
  - By partitioning the state into ‘n’ independent state diagram, only ‘2n’ states are required



Combinatorial explosion of transition in flat state diagram

Expanding States

- One way to organize a model is by having a high-level diagram with sub-diagrams expanding certain states.

  Figure-1 elaborates the dispense state with a lower-level state diagram called a **submachine**.

- A **submachine** is a state diagram that may be invoked as **part of another state diagram**.

CONTD..

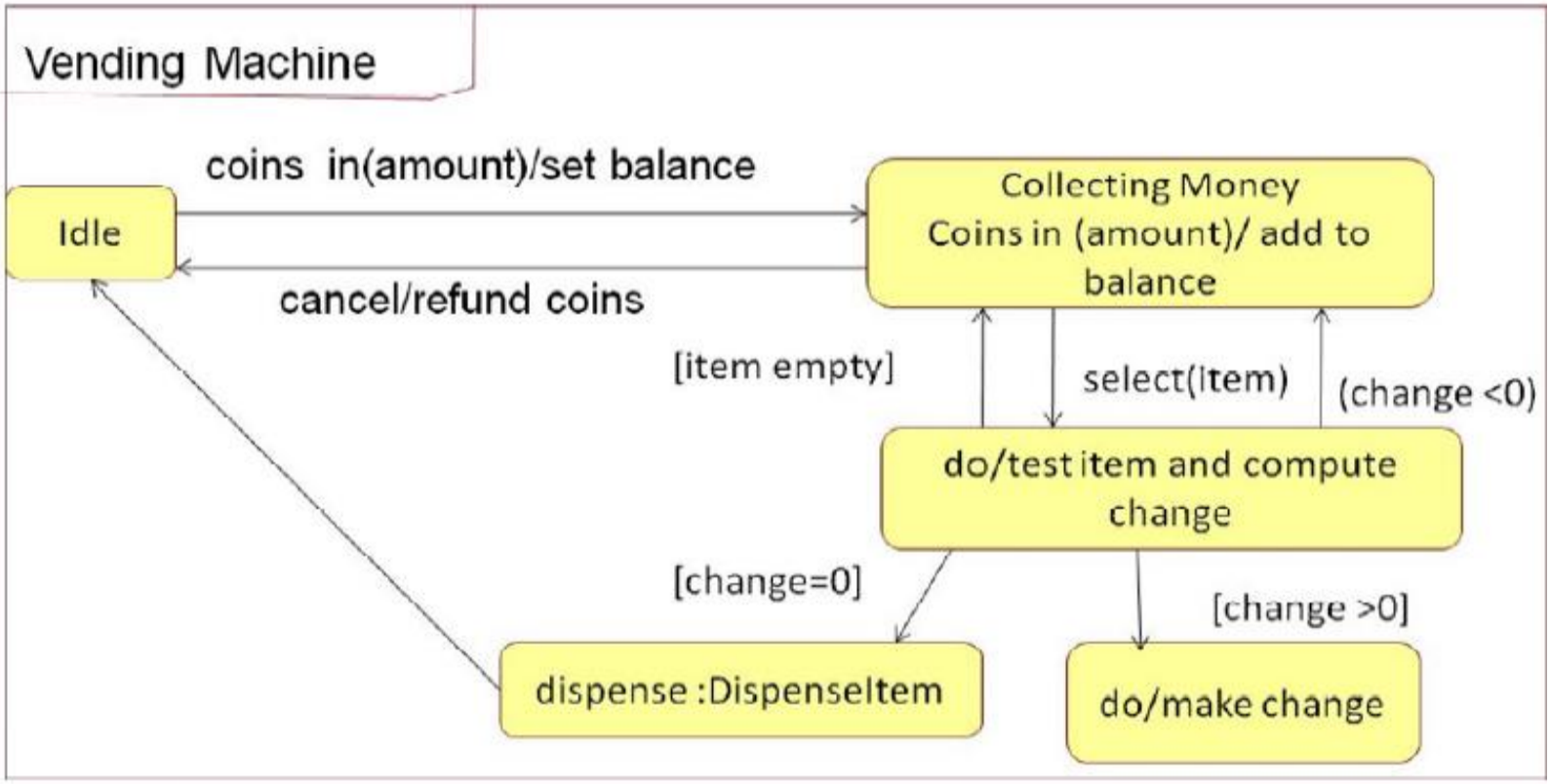


Figure-1: Vending Machine State Diagram

CONTD..

- A submachine is a state diagram that may be invoked as part of another state diagram.

- E.g.: Vending Machine – High –level Diagram, DispenseItem - Subdiagram.
- The submachine replaces the local state.

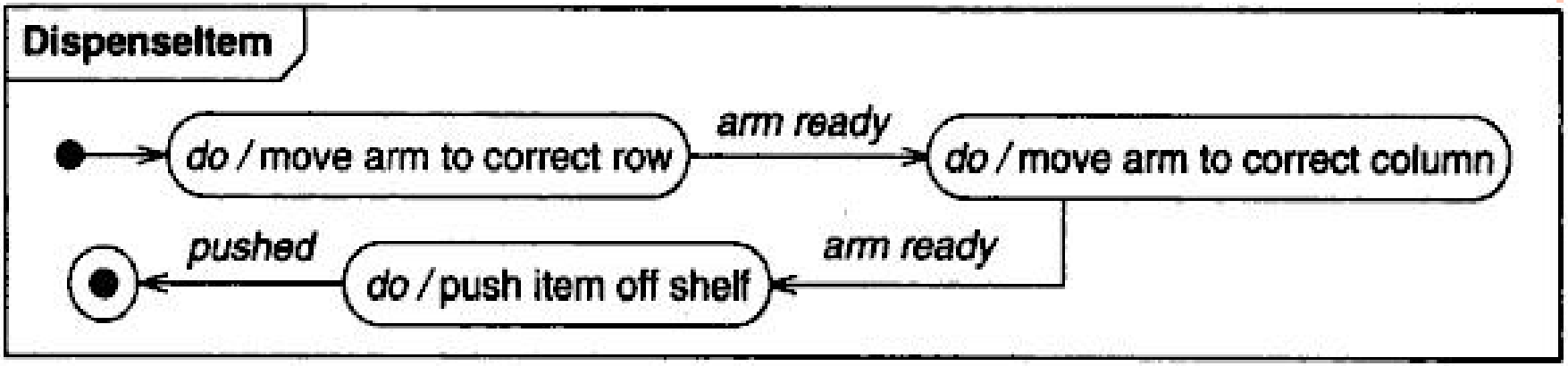
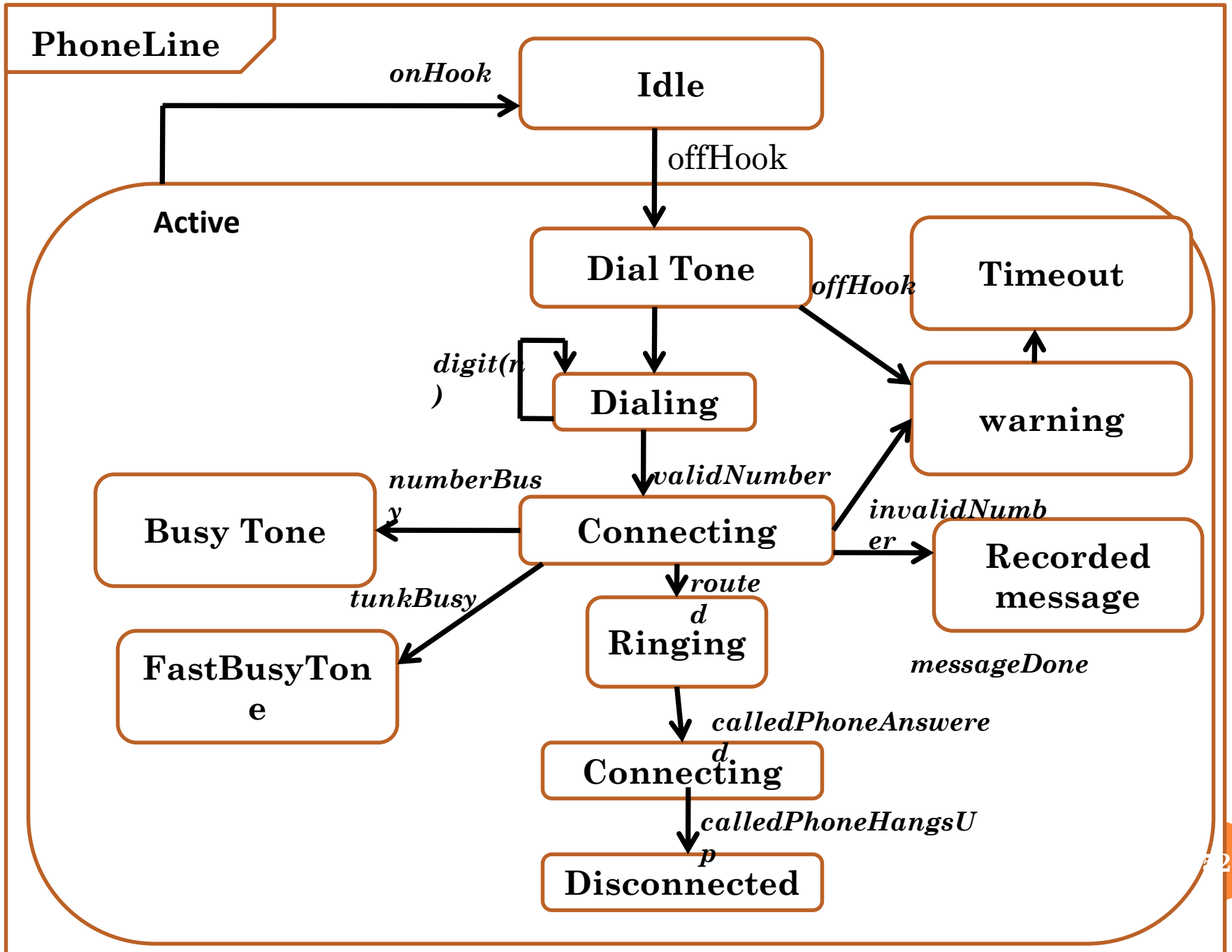


Figure 6.3 ***Dispense item*** **submachine of vending machine.** A lower-level state diagram can elaborate a state.

Nested States

- Second alternative for structuring states is using nested states.

- UML2 avoids using generalization in conjunction with states.
- Local state can be replaced with “nested states”.


PhoneLine
onHook
Idle
offHook
Active
Dial Tone
offHook
Timeout
digit(n)
Dialing
warning
validNumber
numberBusy
Busy Tone
Connecting
invalidNumber
Recorded message
routed
tunkBusy
FastBusyTone
Ringing
messageDone
calledPhoneAnswered
Connecting
calledPhoneHangsUp
Disconnected

CONTD..

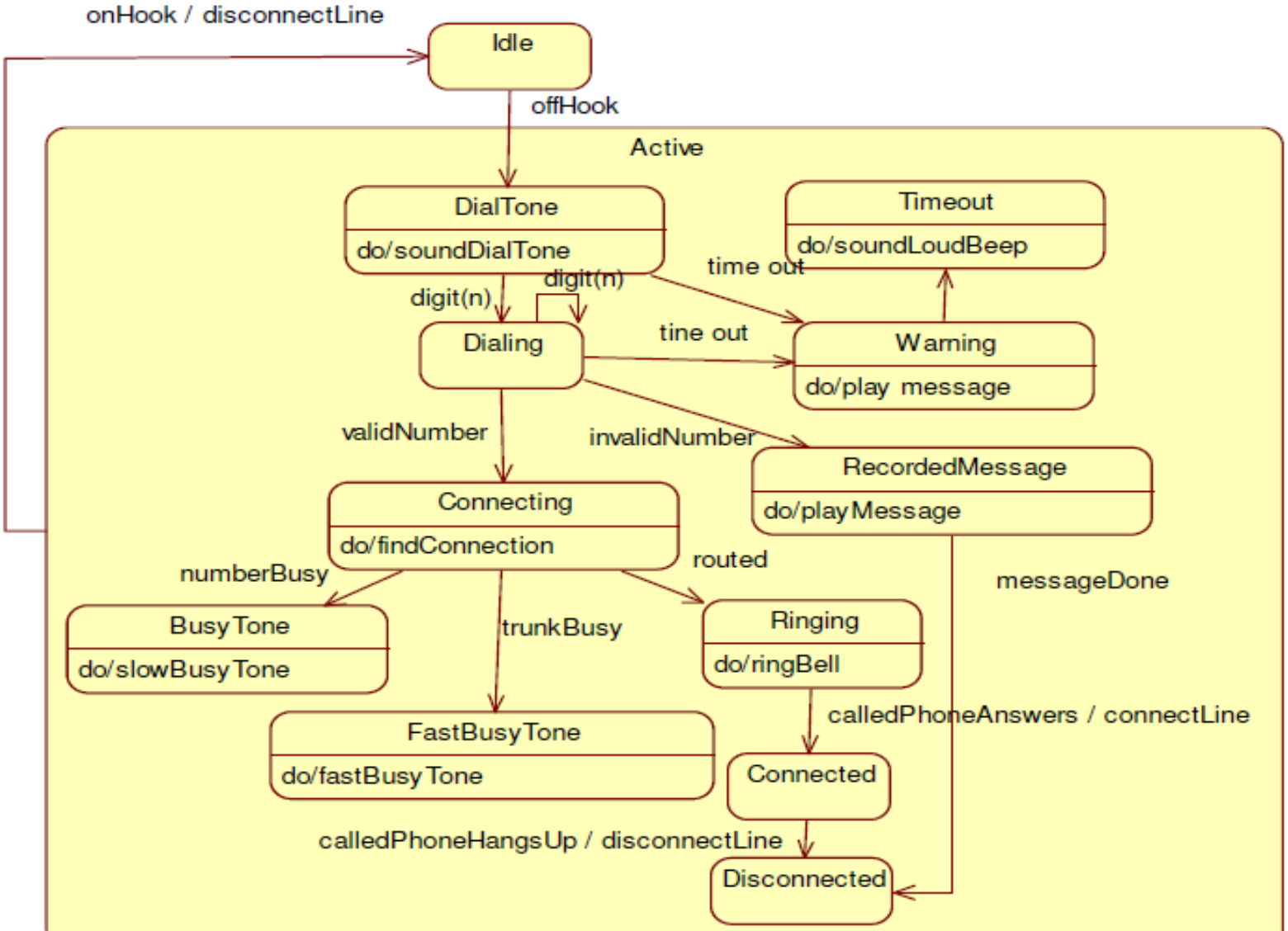


Figure-3: Nested states for phone line

CONTD..

- Figure 3 simplifies the phone line model. The composite state name labels the outer contour that entirely encloses the nested states.
- A nested state receives the outgoing transitions of its composite state. States can be nested to an arbitrary depth (Eg: Car Transmission – Figure 4).

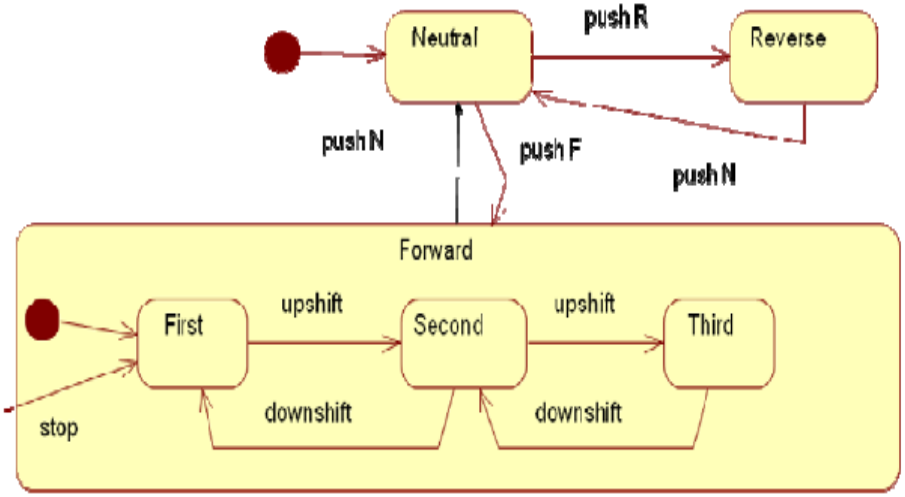


Figure-4: Nested states for Car Transmission

CONTD…

- All three nested states share the transition on event ***stop*** *from the* ***Forward contour*** *to* state ***First***.

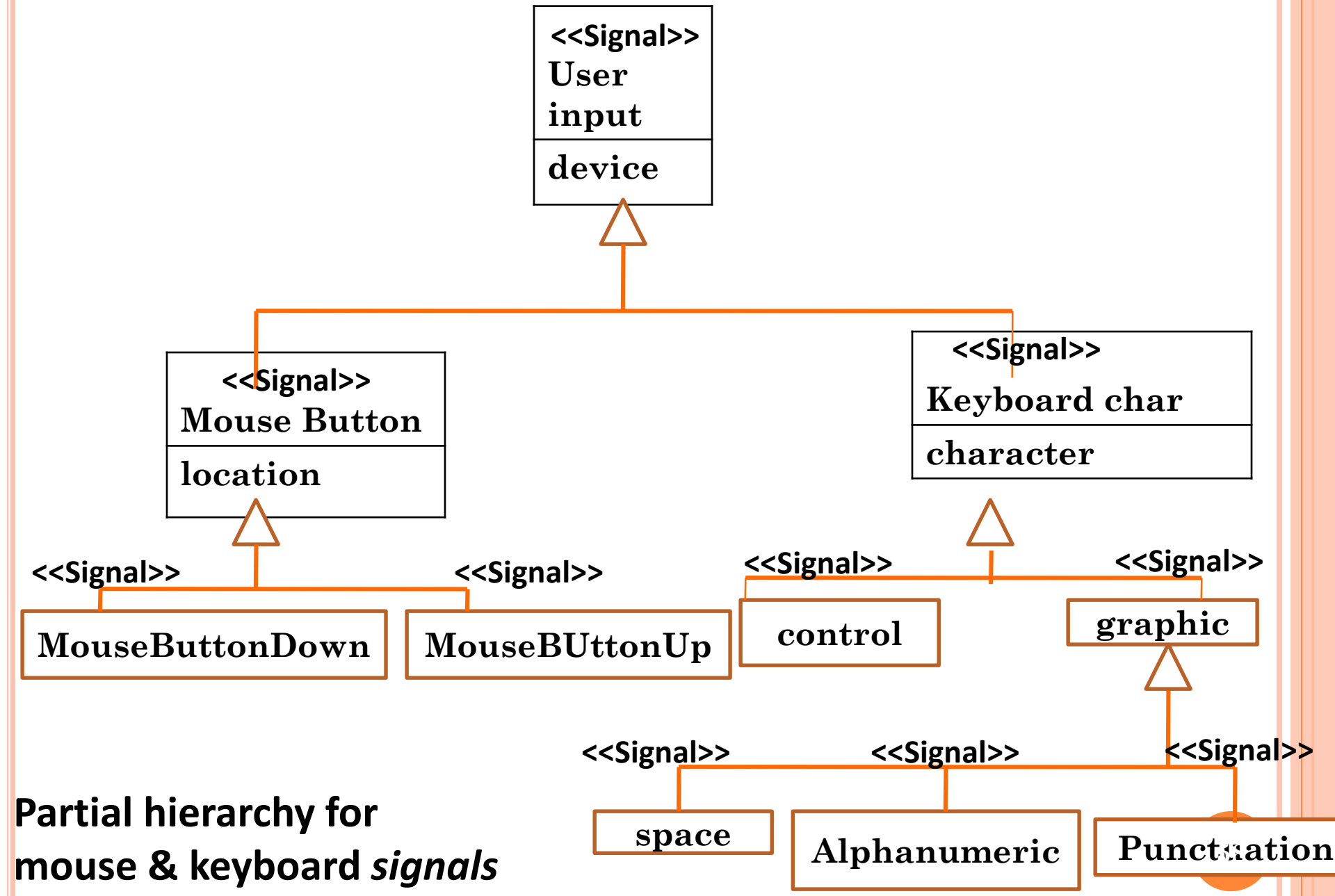
  The nested states can be drawn as separate diagrams and reference them by including a submachine.

CONTD…

- Entry and exit activities are particularly useful in nested state diagrams because they permit a state.
- For a simple problem, implement nested states by degradation into "flat state" diagram.
- The entry activities are executed from the outside in and the exit activities from the inside out.

Signal Generalization

- Signals can be organized into a generalization hierarchy with inheritance of signal attributes.
- It permits different levels of abstraction to be used in a model.
- E.g. Key board signals - Partial hierarchy for keyboard signals.
- The following Figure, shows a part of a tree of input signals for a workstation.



Partial hierarchy for mouse & keyboard *signals*

Concurrency

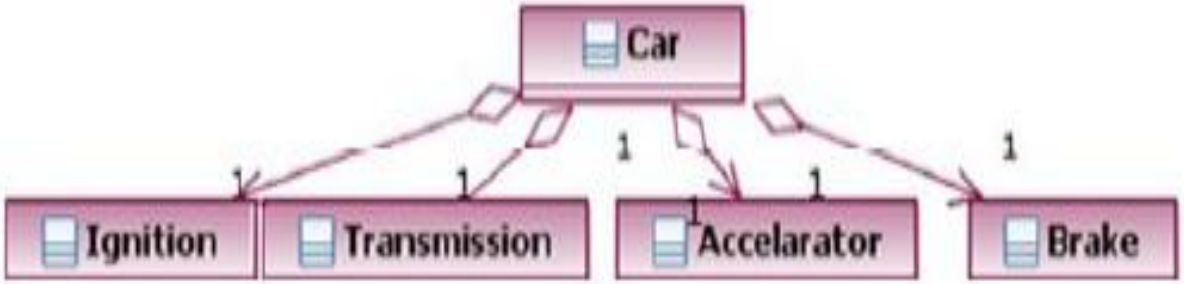
- The state model implicitly supports concurrency among objects.
  - Aggregation Concurrency
  - Concurrency within an Object
  - Synchronization of Concurrent Activities

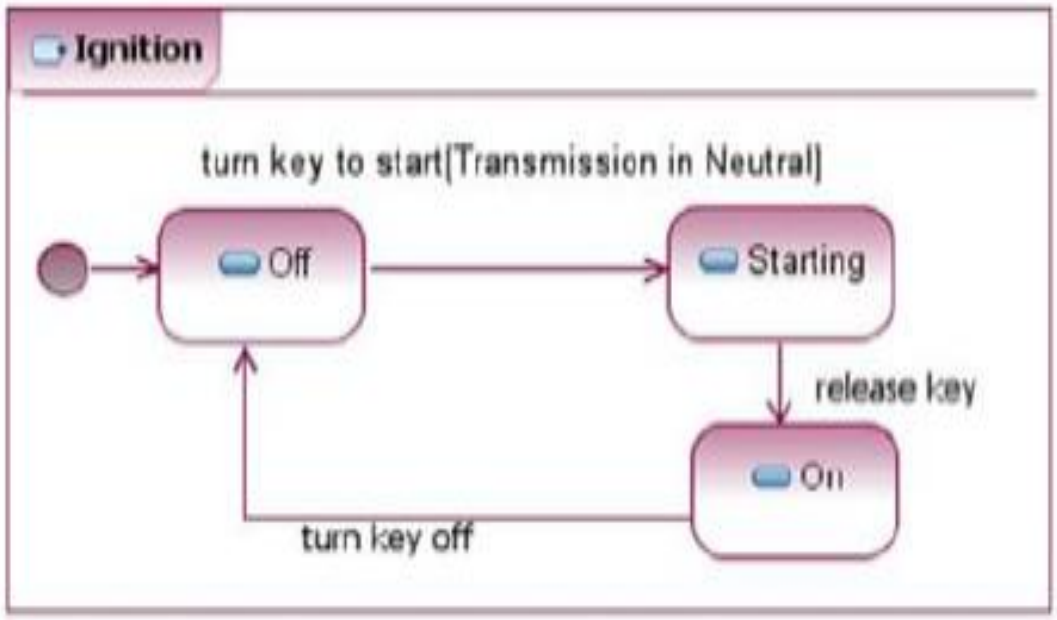
- Relation of Class and State Models

Aggregation Concurrency

- The aggregate state corresponds to the combined states of all the parts. Aggregation is the "and-relationship".
- Transitions for one object can depend on another object being in a given state.
- The following figure shows the state of a *Car as an aggregation of part states*: *Ignition*, *Transmission*, *Accelerator*, *and Brake (plus other unmentioned objects).*

CONTD..


Car
1
1
1
1
1
1
Ignition
Transmission
Accelarator
Brake


Ignition
turn key to start[Transmission in Neutral]
Off
Starting
release key
On
turn key off

CONTD…

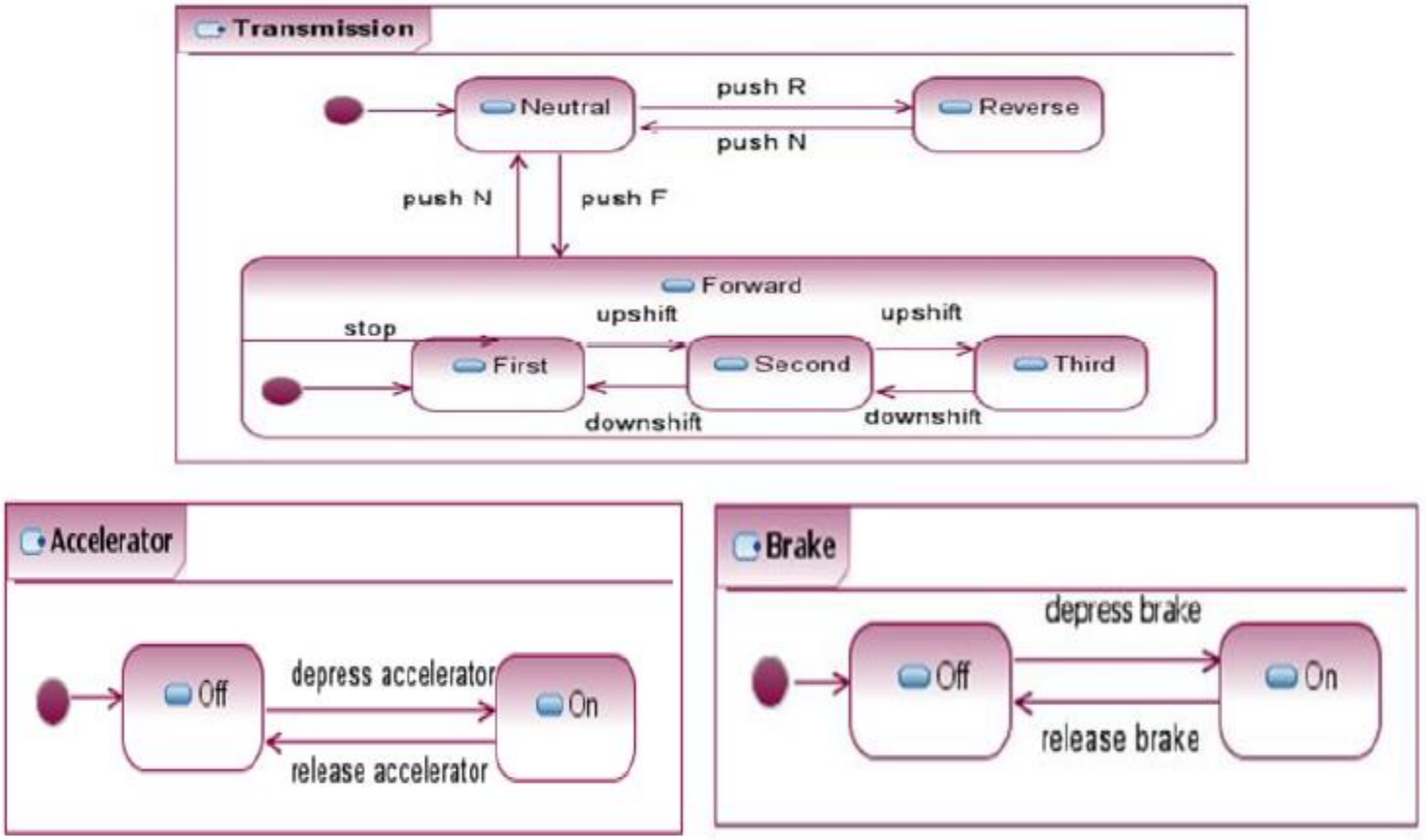


Figure-6: An aggregation and its concurrent state diagrams

CONCURRENCY WITHIN AN OBJECT

Consider an example

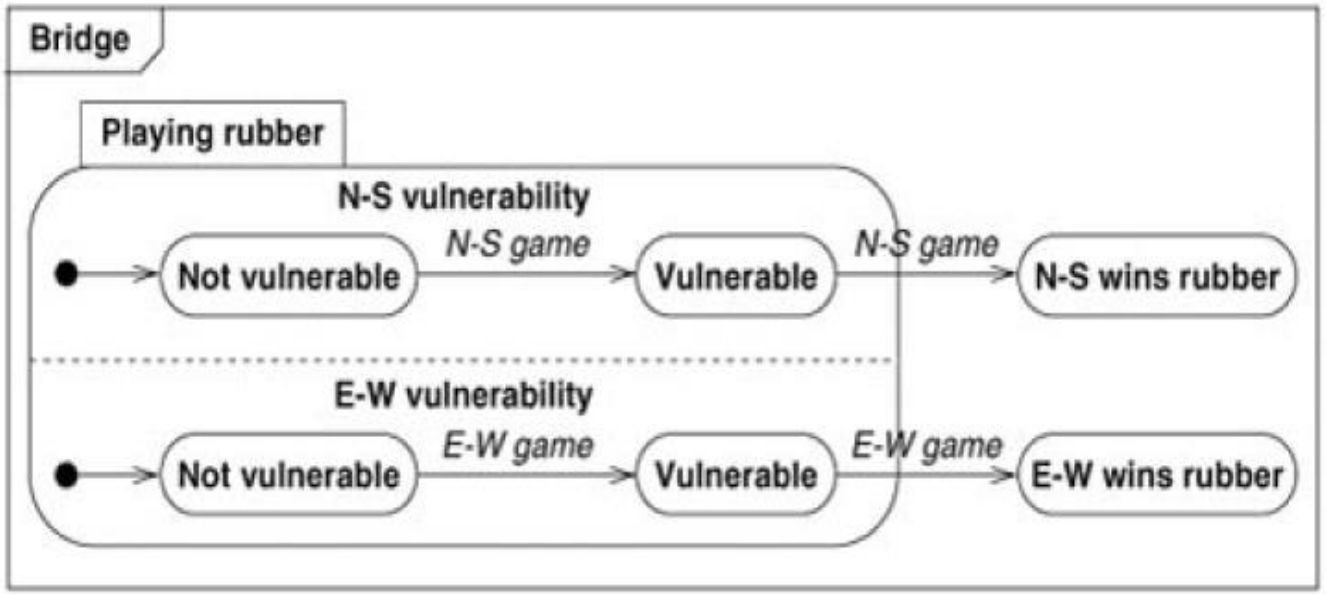


Figure-7: Bridge game with concurrent states

CONTD..

- You can partition some objects into subsets of attributes or links, each of which has its own subdiagram.
- The state of the object comprises one state from each subdiagram. The UML shows concurrency within an object by partitioning the composite state diagram.
- Figure7 shows the state diagram for the play of a bridge rubber. Most programming languages lack intrinsic support for concurrency

Synchronization of Concurrent Activities

- The object does not synchronize the internal steps of the activities but **must complete both activities** before it can progress to its next state.

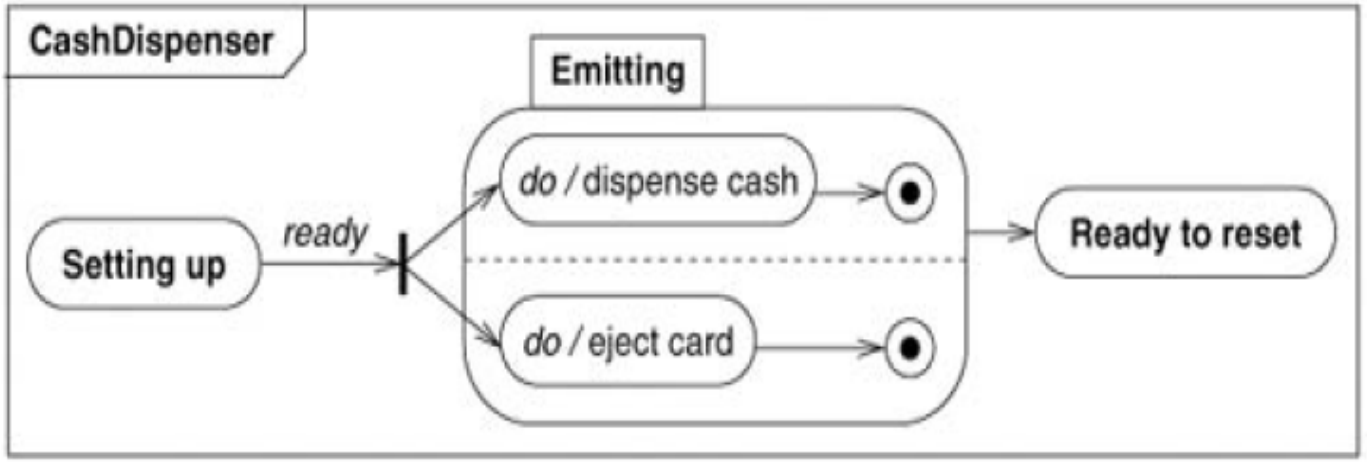


Figure-8: Synchronization of control

CONTD..

- A cash dispensing machine dispenses cash and returns the user's card at the end of a transaction.
- Each region is a sub diagram that represents a concurrent activity within the composite activity.
- If any sub diagram in the composite state are not part of the merge, they automatically terminate when the merge transition fires.

CONTD..

- A completion transition fires when activity in the source state is complete.

- The firing of a merge transition causes a state diagram to perform the exit activities (if any) of all subdiagrams, in the case of both explicit and implicit merges.

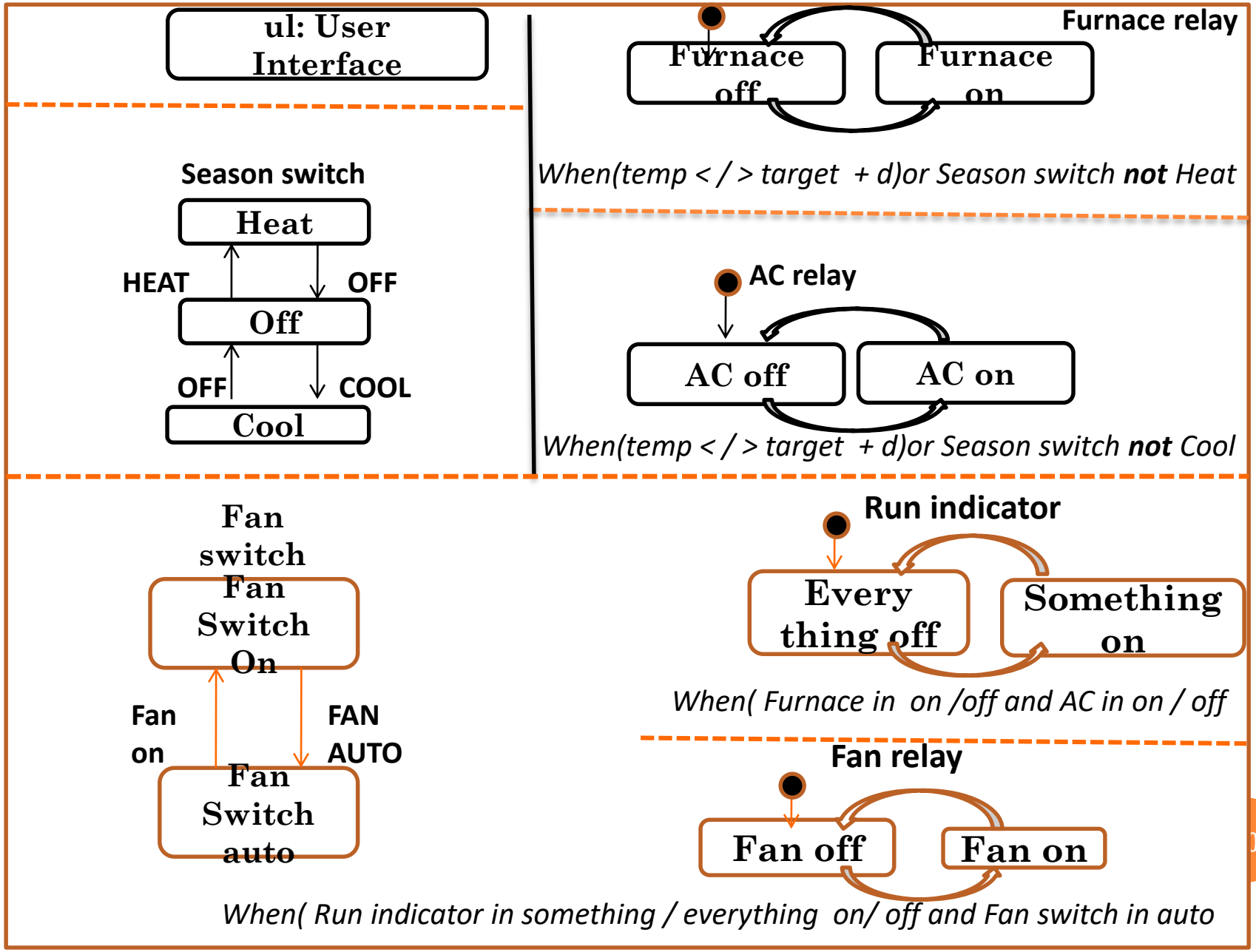
Relation of Class and State Models

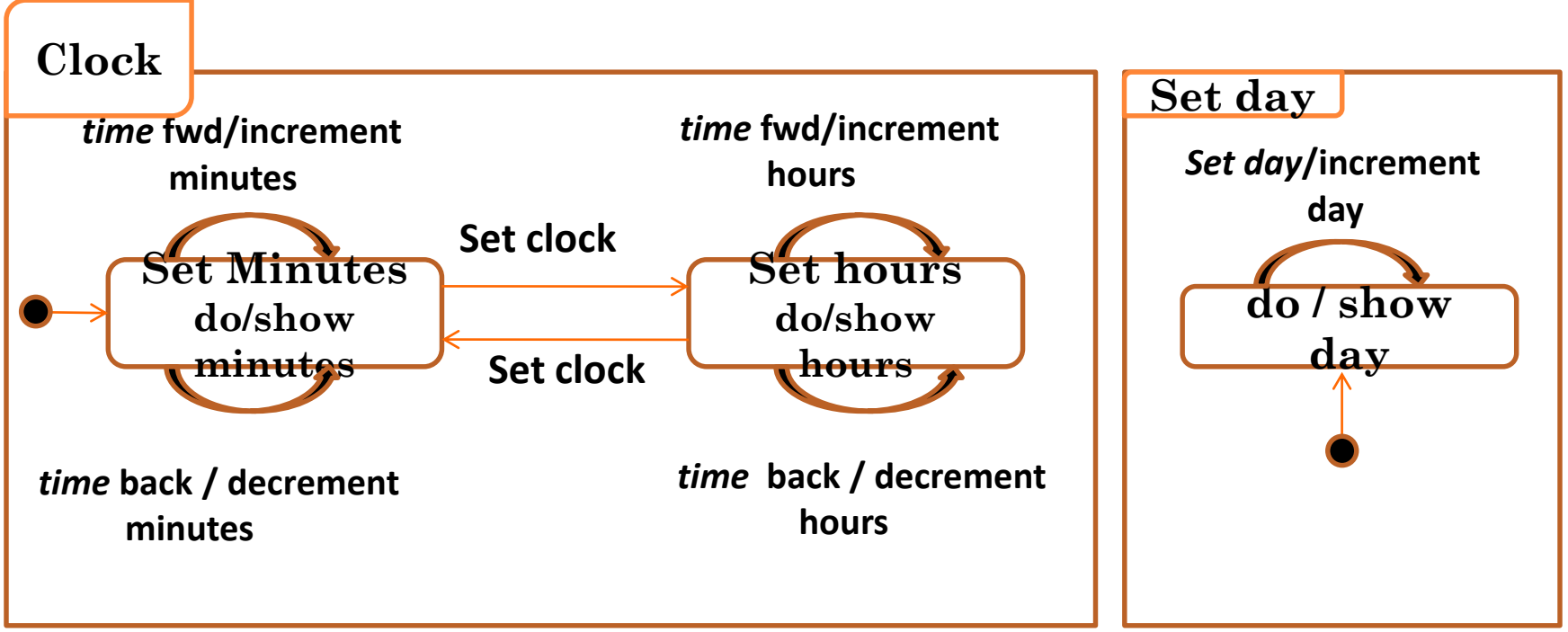
- A state diagram describes all or part of the behaviour of the objects of a given class.
- A single object can have different states over time – the object preserves its identity – but it cannot have different classes.
- There are three sources of concurrency within the class model:
  - Aggregation of objects
  - Aggregation within an object
  - Concurrent behaviour of an object

CONTD..

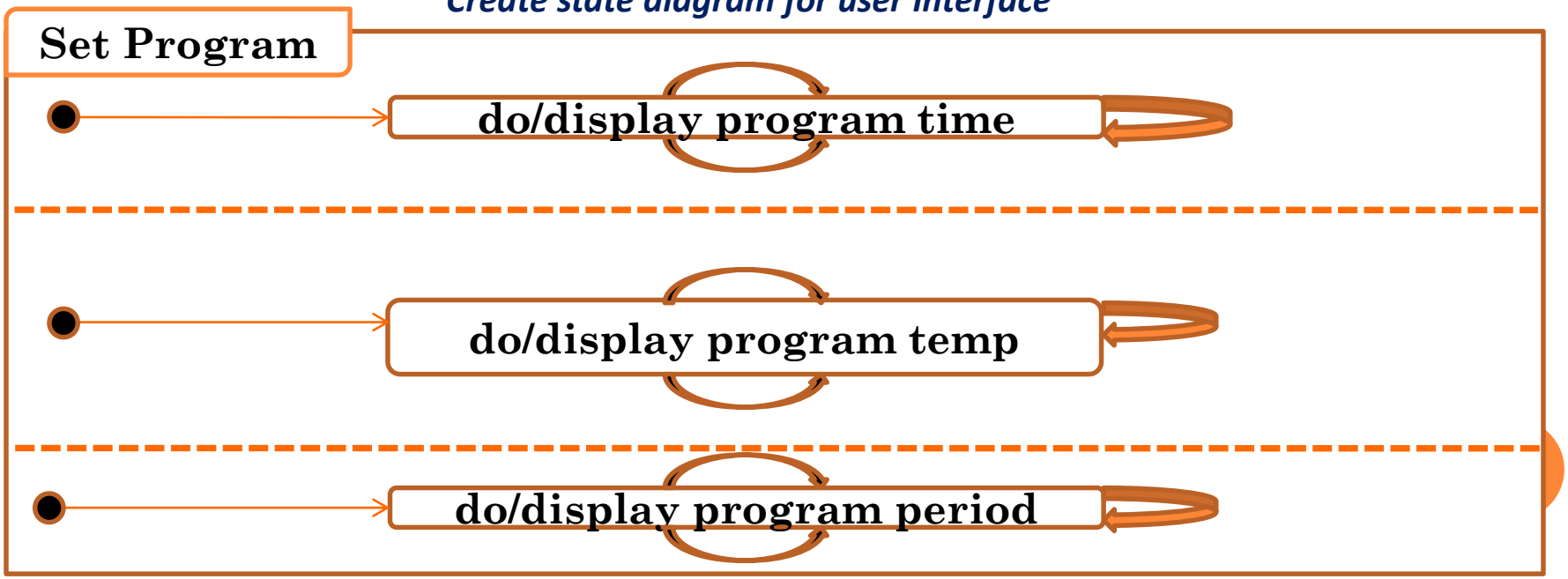
- Aggregation within an object: the values and links of an object are its parts, and groups of them taken together define concurrent substates of the composite object state.
- The state model of a class is inherited by its subclasses. The state diagram of the subclass must be a refinement of the state diagram of the superclass.
- Transitions can often be implemented as operations on objects.

A SAMPLE STATE MODEL [Programmable Thermostat]

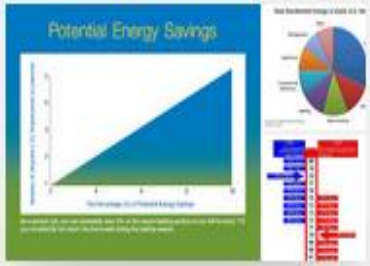




Create state diagram for user Interface



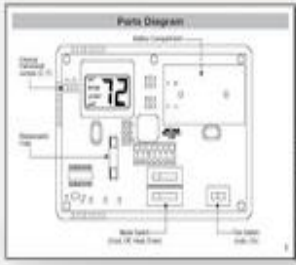

Vaillant
22.5
12:2
i
F
P
colorMATIC



Savings

Honeywell

Energy Star



Wiring

End of State Modelling