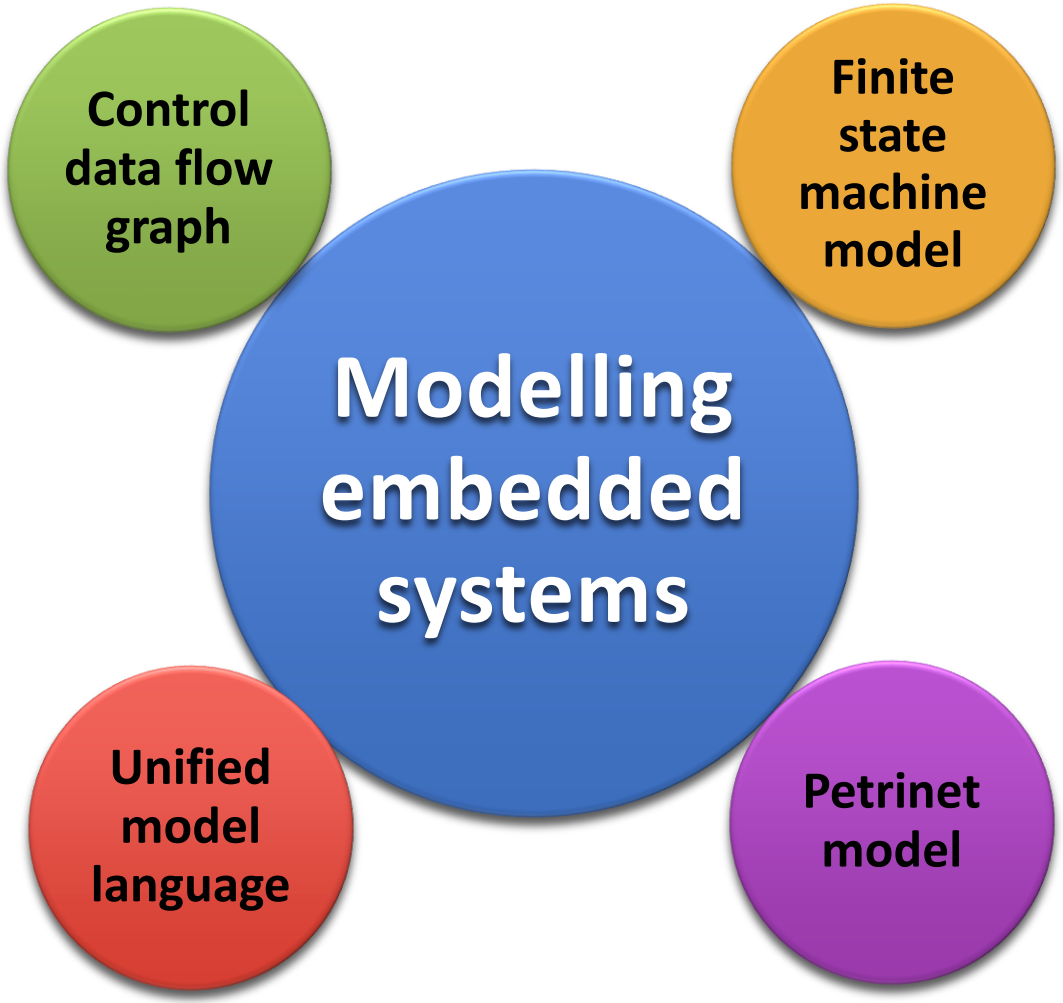
BECE403E-EMBEDDED SYSTEM DESIGN

MODULE-4

MODELLING EMBEDDED SYSTEMS

MODULE-4


Control data flow graph
Finite state machine model
Modelling embedded systems
Unified model language
Petrinet model

INTRODUCTION

INTRODUCTION

- Modelling in ES refers to the process of creating abstract representations of the hardware, software, and the interactions between them within an embedded system.
- Modelling helps in the initial design phase by providing a clear understanding of the system's architecture, functionality, and interactions.
- Importance of modelling : Visualization, reduce complexity, documentation
- Modelling processes are used in ES specifically for software analysis and design before software implementation.
- A software analysis and design helps
  - A description of the system requirement
  - A description of how system works
  - Shows system validation

INTRODUCTION

- To effectively model a system, you need a language with which the model can be described
- Modelling language is a graphical/textual computer language explains design and construction of any model or structure following a set of guidelines.
- There are various modelling techniques and tools available for embedded systems, including,
  1. Control Data Flow Graph (CDFG) Model
  2. Finite States Machine(FSM) Model
  3. Petri Net Model
  4. Unified Modelling Language (UML)

UNIFIED MODELLING LANGUAGE (UML)

UNIFIED MODELLING LANGUAGE

- UML - general purpose visual modelling language used to visualize, specify, construct, and document

- It can be used for both modelling the software system and non-software systems

- UML can be effectively used for modelling embedded systems for better understanding of system requirements, architecture, behaviour, and deployment, facilitating communication, design, and implementation processes.

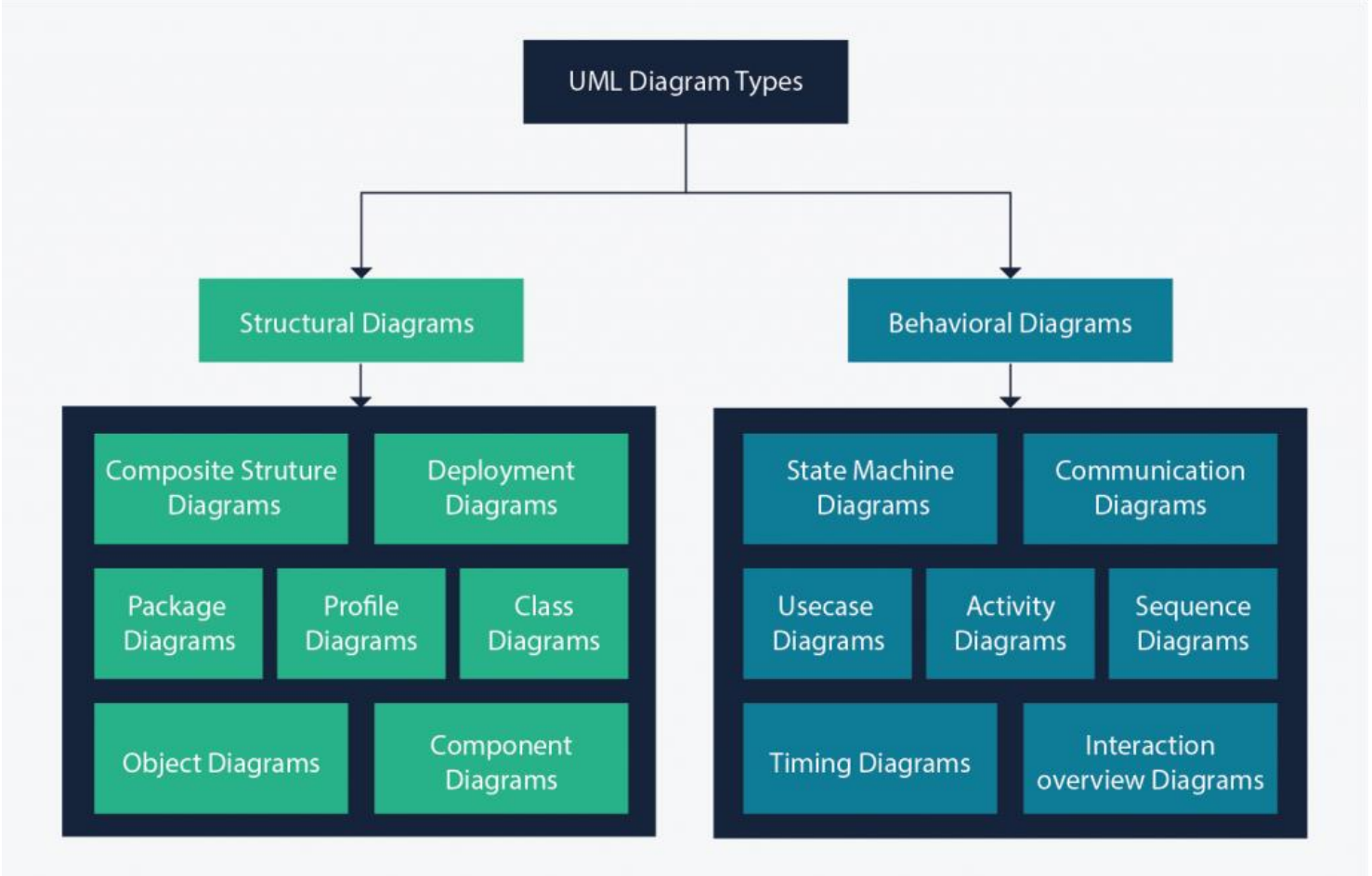
- UML Uses:
  - ✓ Forecast systems
  - ✓ To estimate the reusability
  - ✓ Plan and analyse system behaviour
  - ✓ Easier maintenance/modification

UNIFIED MODELLING LANGUAGE

- Two types of diagram in UML

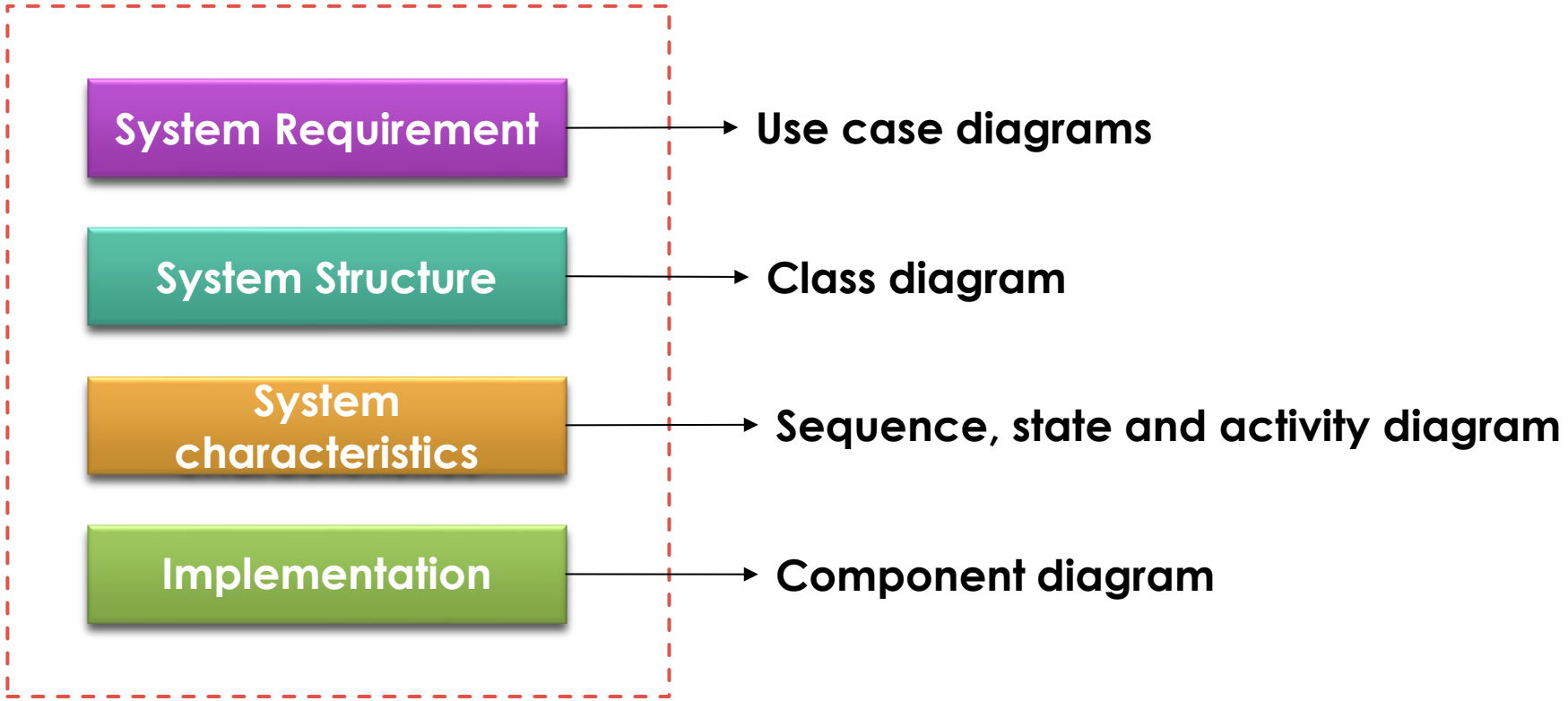
- Structure diagrams show the things in the modelled system. In a more technical term, they show different objects in a system.

- Behavioural diagrams show what should happen in a system. They describe how the objects interact with each other to create a functioning system.



UNIFIED MODELLING LANGUAGE

- Principles of UML diagram



UNIFIED MODELLING LANGUAGE

USE CASE DIAGRAM

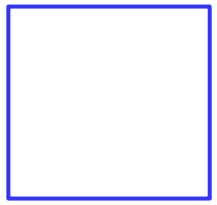
- A use case diagram is used to represent the dynamic behaviour of a system, it summarize the details of your system's users and their interactions with the system.

- Use case diagrams model the functionality of a system using actors and use cases.
  - Use cases are a set of actions, services, and functions that the system needs to perform.
  - "system" is something being developed or operated
  - "actors" are people or entities operating under defined roles within the system.

- UML use case diagrams are ideal for:
  - Representing the goals of system-user interactions
  - Defining and organizing functional requirements in a system
  - Specifying the context and requirements of a system
  - Modelling the basic flow of events in a use case

UNIFIED MODELLING LANGUAGE

USE CASE DIAGRAM - NOTATIONS

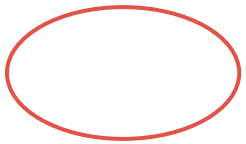


System: A specific sequence of actions and interactions between actors and the system.
Software component, Process, Application etc.,

Actors: The users that interact with a system.

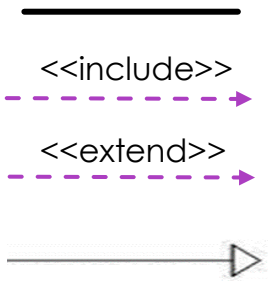
(1) Primary Actor-Initiate the use of the system

(2) Secondary Actor - Reactionary



Use Cases: Represents an action that accomplishes or some sort of a task within the system

Relationships:



(1) Association – basic communication or interaction

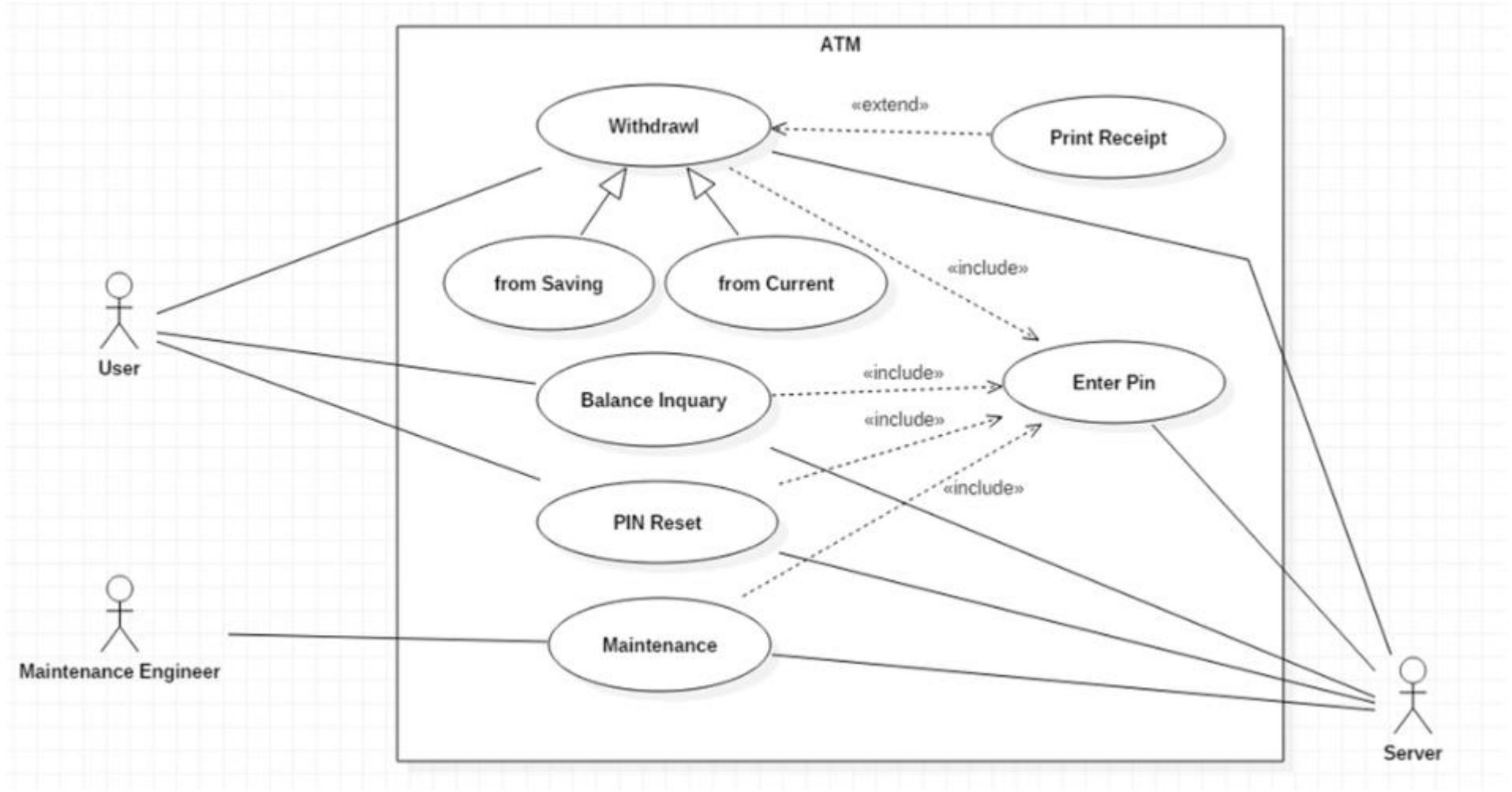
(2) Include – Base use case require include use case in order to be complete

(3) Extend - Base use case require extend use in some time but not every time or happen only when certain criteria is met

(4) Generalization – represents parent – child relationship

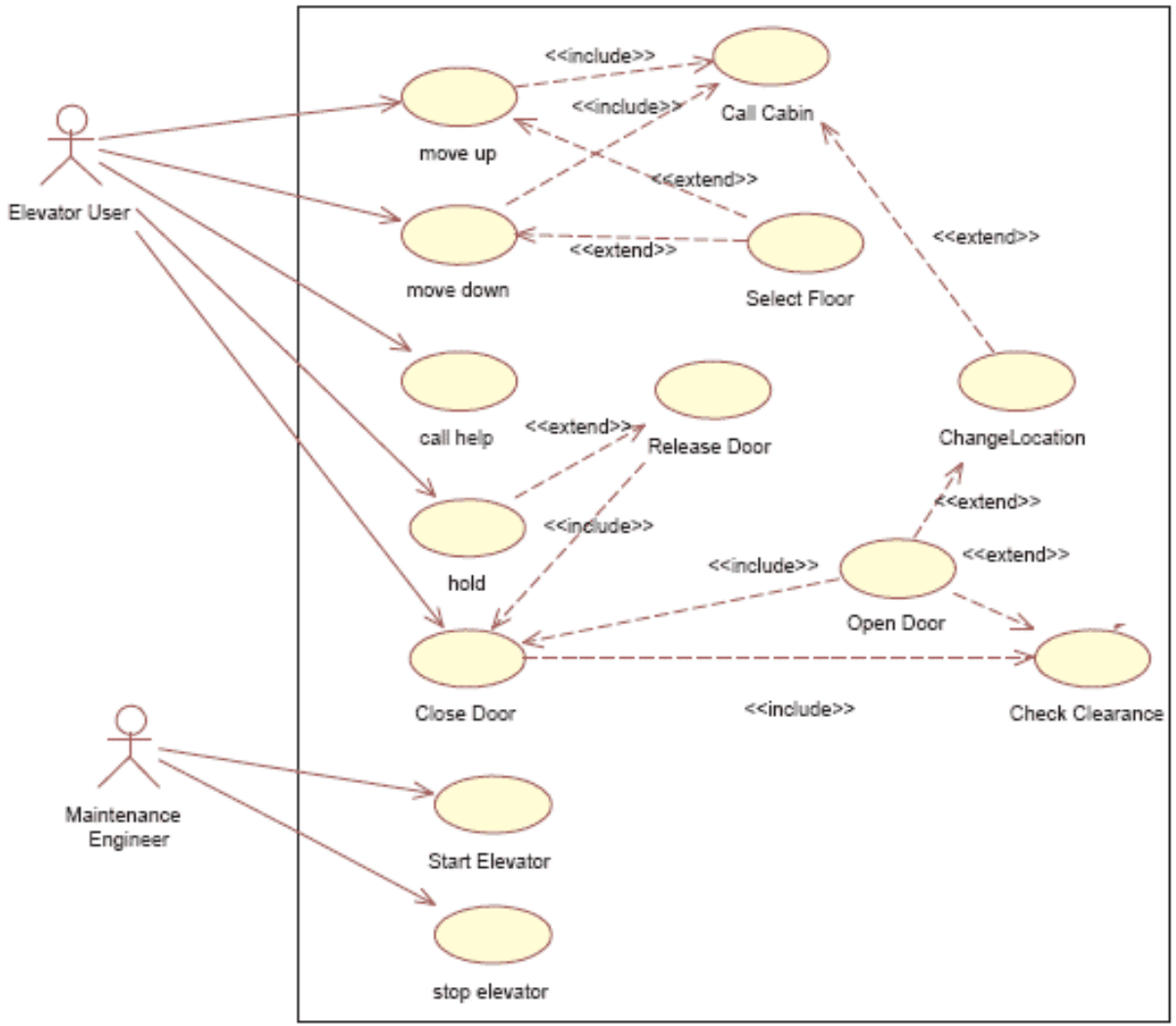
UNIFIED MODELLING LANGUAGE

USE CASE DIAGRAM EXAMPLE – ATM MACHINE



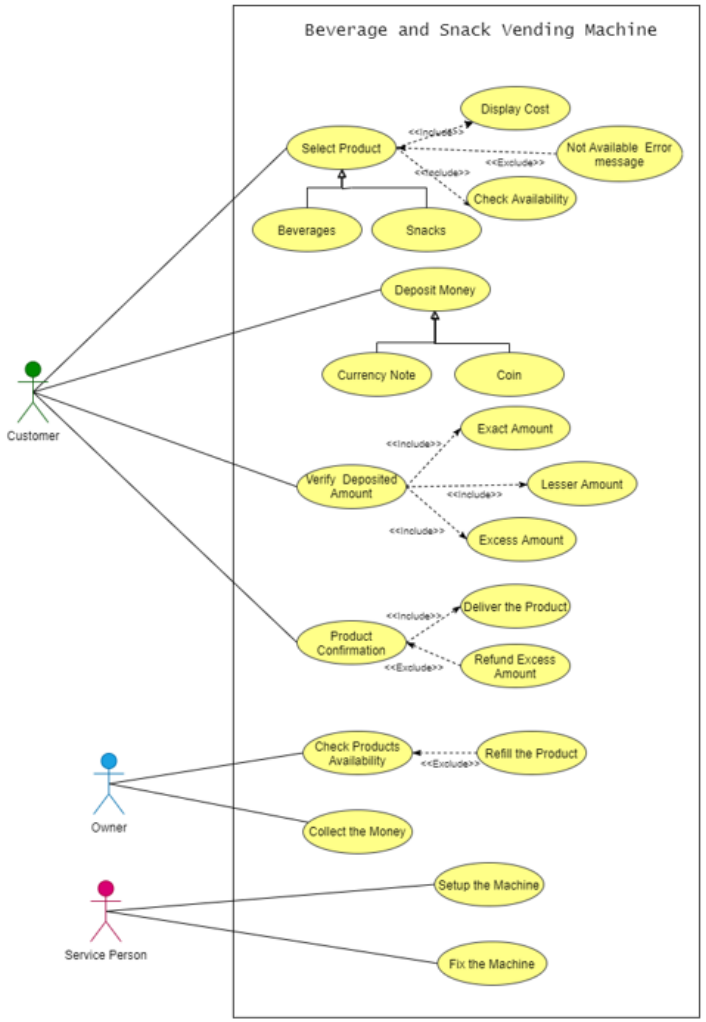
UNIFIED MODELLING LANGUAGE

USE CASE DIAGRAM EXAMPLE – ELEVATOR SYSTEM



UNIFIED MODELLING LANGUAGE

USE CASE DIAGRAM EXAMPLE – VENDING MACHINE



UNIFIED MODELLING LANGUAGE

SEQUENCE DIAGRAM

- A UML sequence diagram is a type of interaction diagram that illustrates how objects interact in a particular scenario of a system.
- It depicts the sequence of messages exchanged between objects or components over time to accomplish a specific task or scenario.
- Sequence diagrams are particularly useful for visualizing the dynamic behavior of a system and understanding the flow of control between different elements.
- Sequence diagram shows the details of the use cases, that can used in planning and understanding the functionality of existing and future scenario
- Time is represented in vertical direction and the Header elements are represented in Horizontal direction

UNIFIED MODELLING LANGUAGE

SEQUENCE DIAGRAM - NOTATIONS

Actor

Object

Lifeline Notation

Activation Bar

Synchronous message

Return message

Comment

Alternate frame

Alt

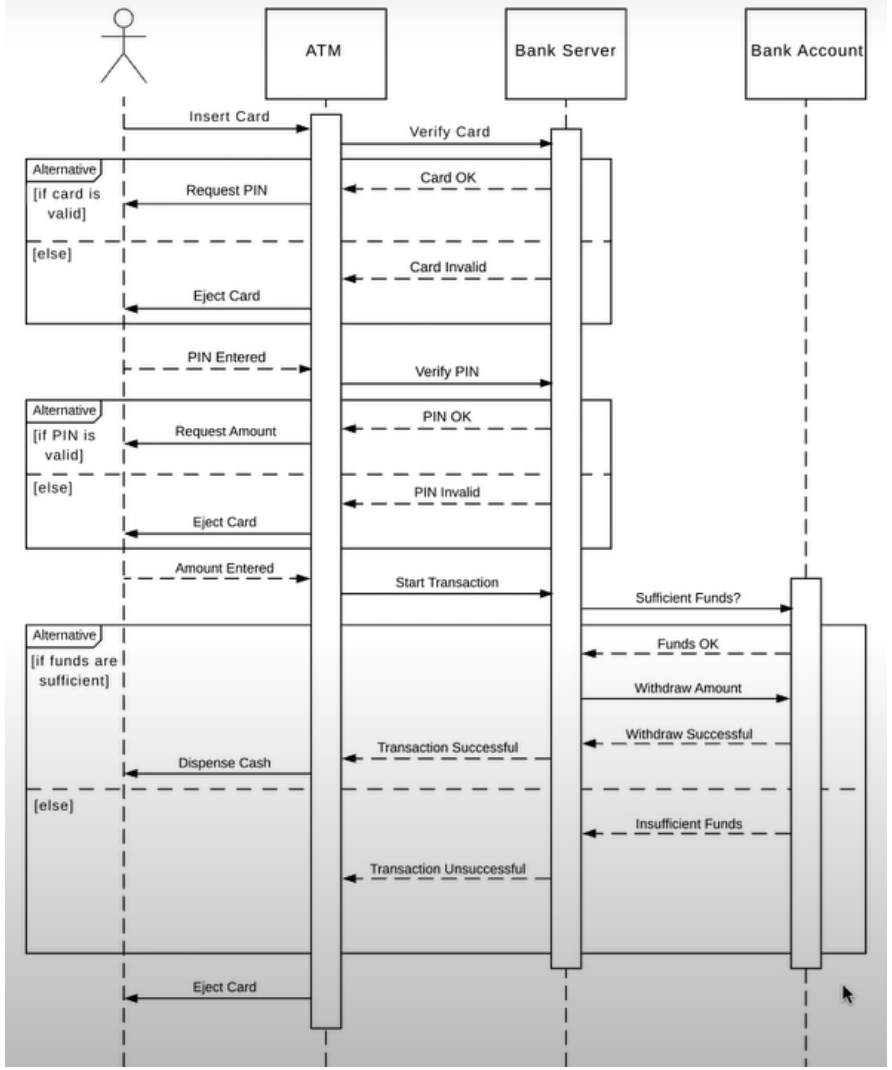
If it is valid

Else

Asynchronous message

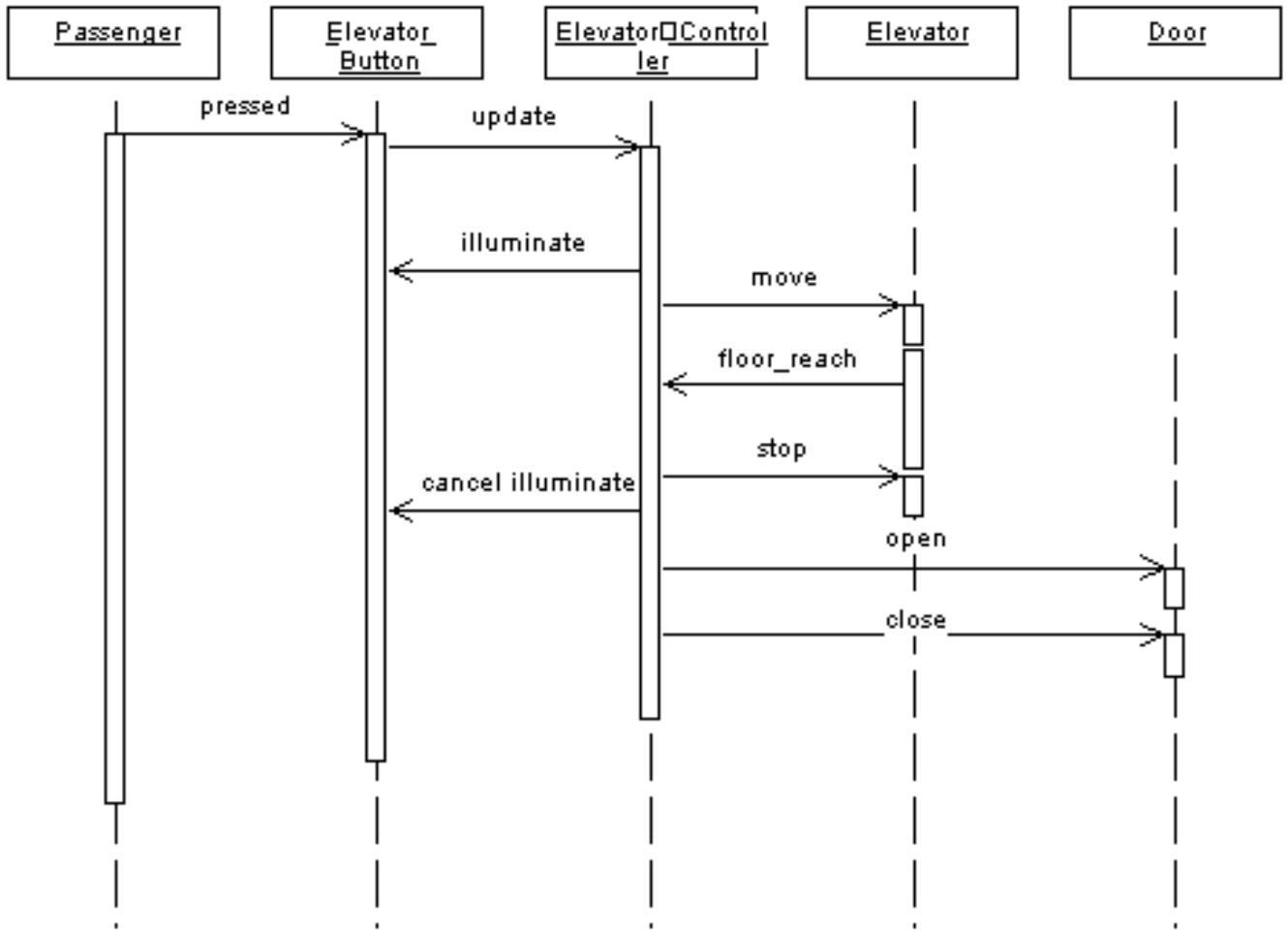
UNIFIED MODELLING LANGUAGE

SEQUENCE DIAGRAM EXAMPLE – ATM MACHINE



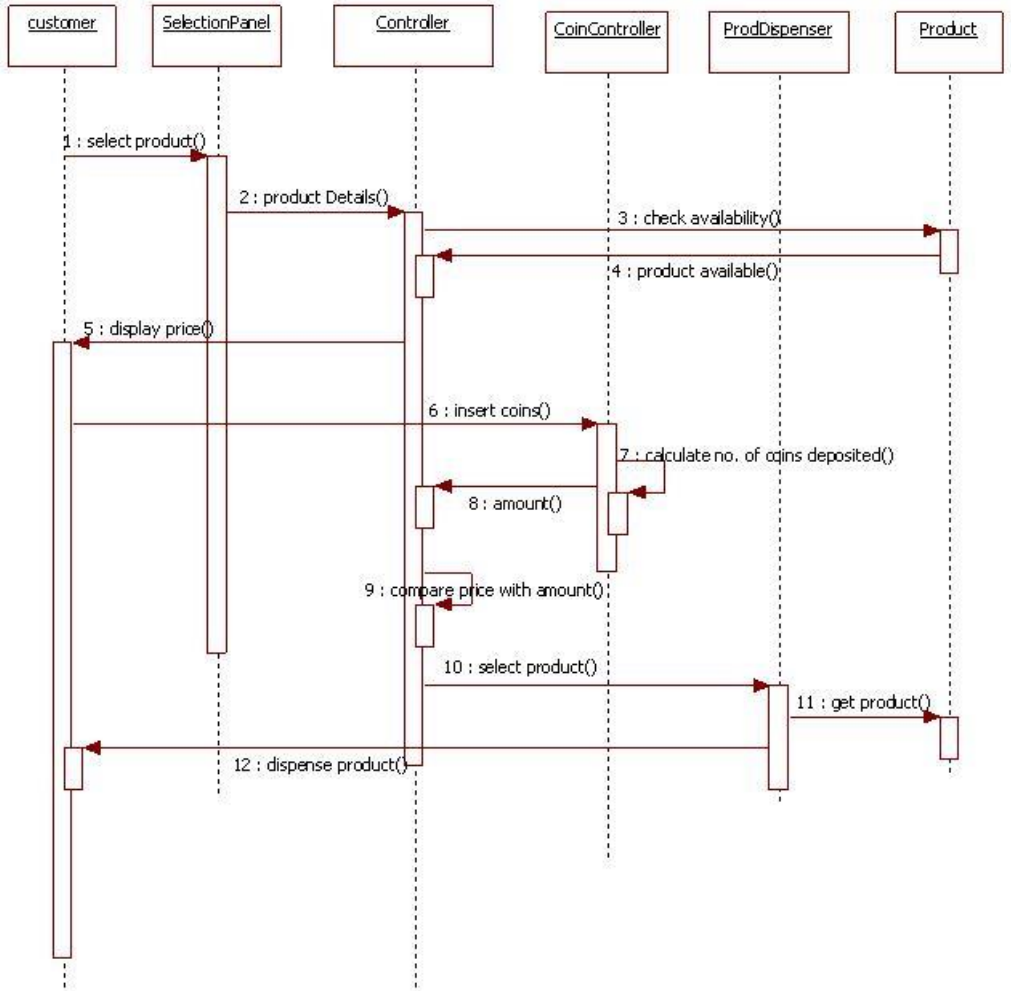
UNIFIED MODELLING LANGUAGE

SEQUENCE DIAGRAM EXAMPLE – ELEVATOR SYSTEM



UNIFIED MODELLING LANGUAGE

SEQUENCE DIAGRAM EXAMPLE - VENDING MACHINE



UNIFIED MODELLING LANGUAGE

CLASS DIAGRAM

- Class diagram is a static type structure diagram that describes the structure of the system

- Creating a class diagram for an embedded system involves identifying the key classes, their attributes, methods, and relationships that form the structure of the system.

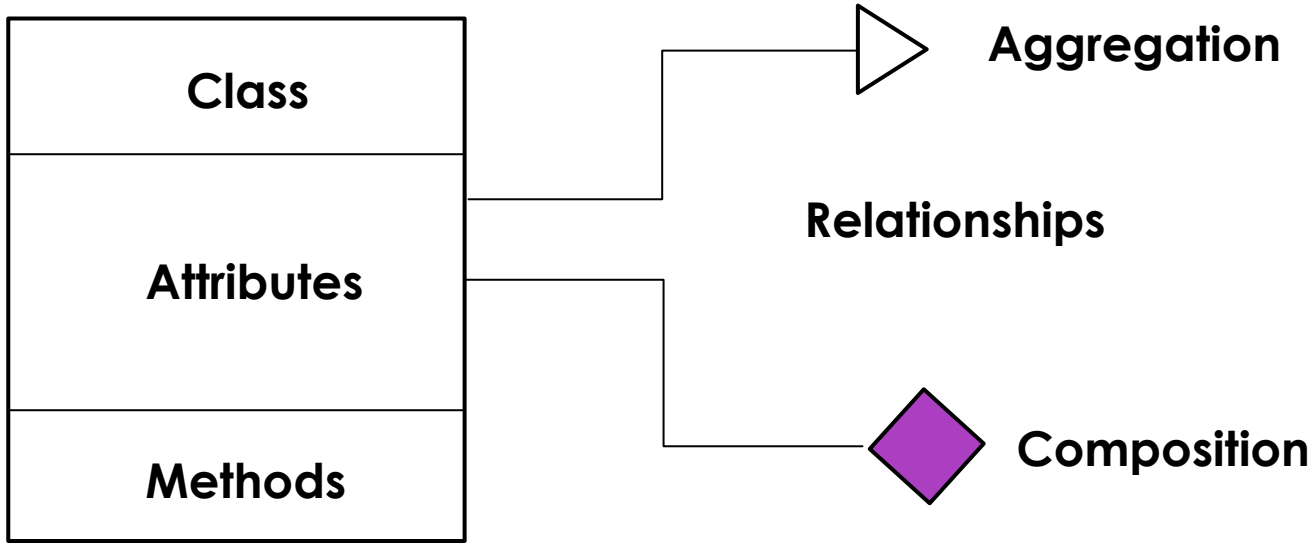
- In an embedded system, classes often represent hardware components, software modules, interfaces, and other relevant entities

- The purpose of the class diagram can be summarized as –
  - Design and Analyze of the system
  - Describe responsibilities of a system
  - Conceptual modelling
  - Forward and reverse engineering

UNIFIED MODELLING LANGUAGE

CLASS DIAGRAM

- UML class diagram is divided into four parts.
  - Top section - name the class.
  - Second - attributes of the class(fields, variables, properties).
  - Third - operations or methods performed by the class.
  - Fourth - any additional components.



UNIFIED MODELLING LANGUAGE

CLASS DIAGRAM - ATTRIBUTES

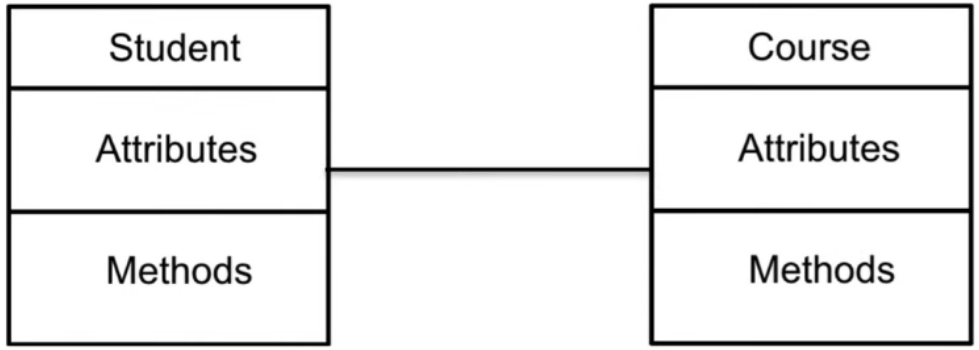
- Attributes are the type of data items that objects of the class will store.
- Attribute also known as variables, properties or fields
- Attributes visibility
  - + Public
  - - Private
  - # protected
  - ~ Package/default
- Attributes visibility – Access rights

Access Right	public (+)	private (-)	protected (#)	Package (~)
Members of the same class	yes	yes	yes	yes
Members of derived classes	yes	no	yes	yes
Members of any other class	yes	no	no	in same package

UNIFIED MODELLING LANGUAGE

CLASS DIAGRAM - NOTATIONS

- **Association:** Shows that the two classes able to communicate with each other.



- **Multiplicity:** Indication of how many objects may participate

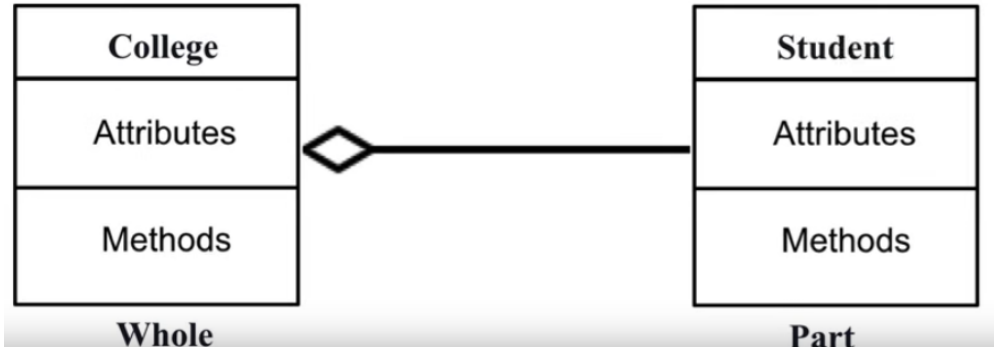
Multiplicity

0..1	zero to one (optional)
n	specific number
0..*	zero to many
1..*	one to many
m..n	specific number range

UNIFIED MODELLING LANGUAGE

CLASS DIAGRAM - NOTATIONS

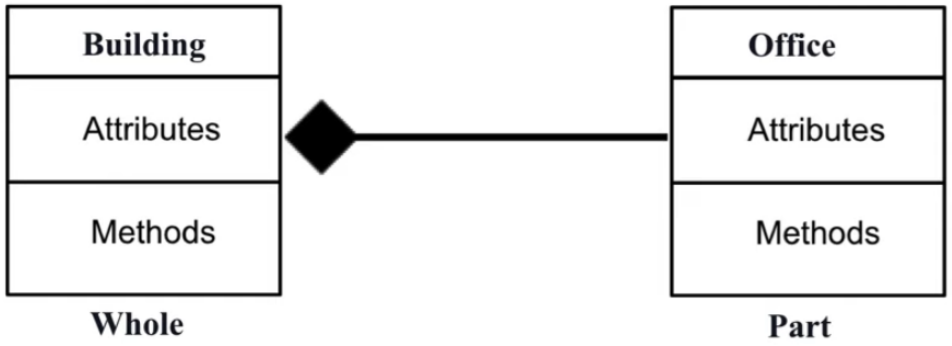
- Aggregation is a special type of "Whole-Part" association between two classes where the "part" can exist separately from "whole".



Other Examples:

- Library and books
- Employee list and Employee
- Hospital and Doctor
- Team and Player

- Composition is a special type of "Whole-Part" association between two classes where the "part" cannot exist separately from "whole".



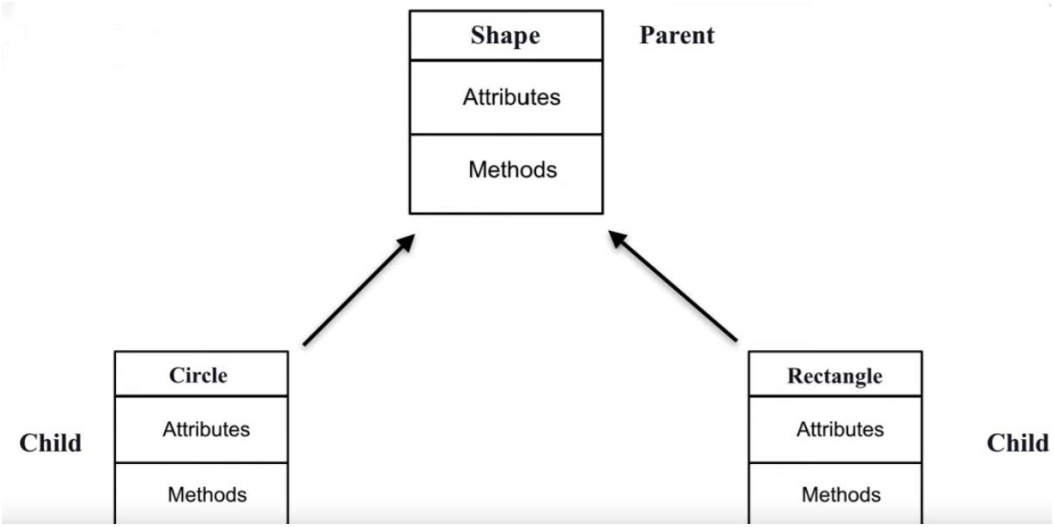
Other Examples:

- Browser and Tab
- Person and head
- Pond and fish
- Folder and file

UNIFIED MODELLING LANGUAGE

CLASS DIAGRAM - NOTATIONS

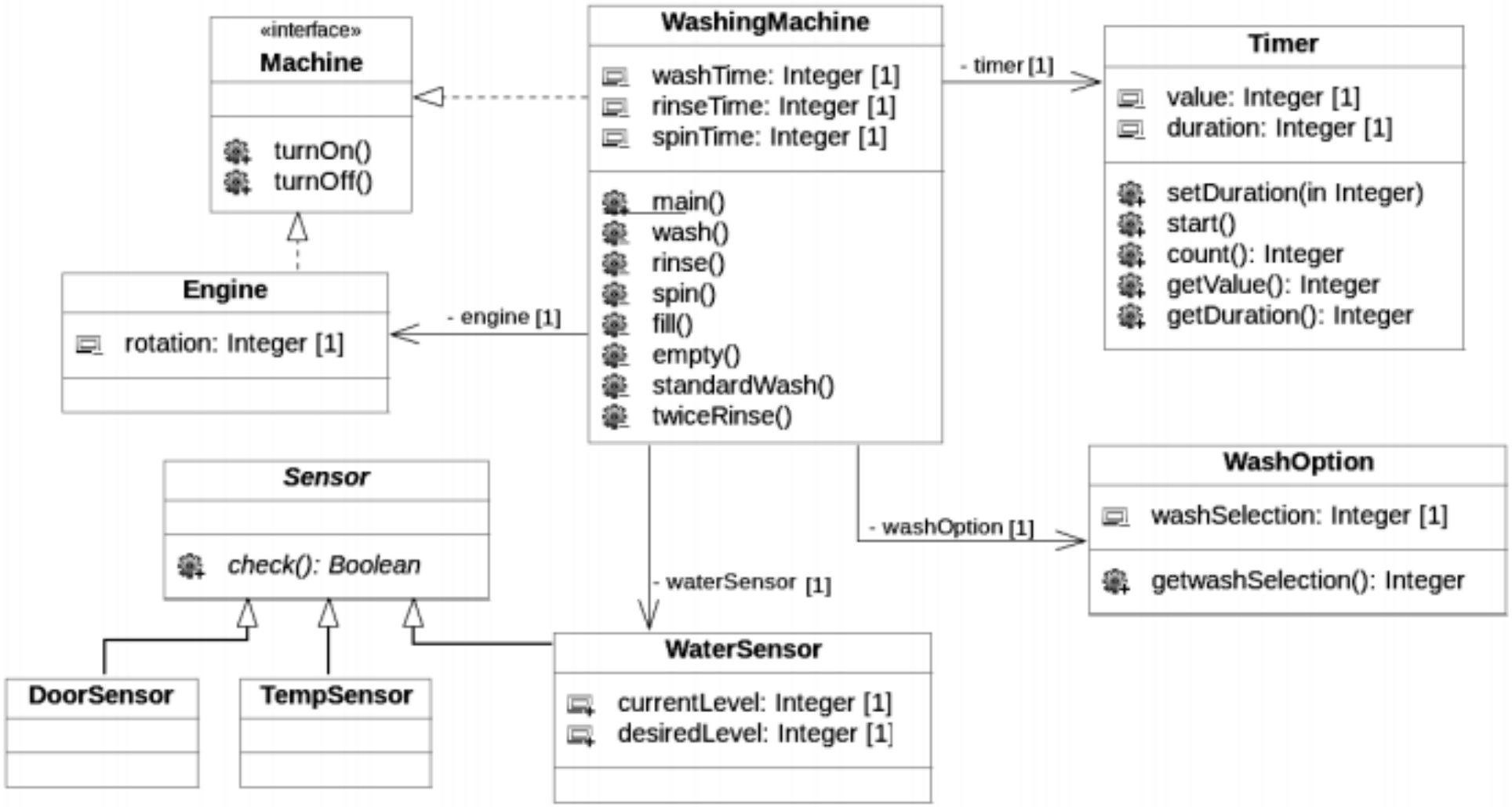
- **Inheritance:** Showing parent – Child relationship



- **Dependency Relationship:** If the changes to the definition of one may cause a change in the other but not the vice versa.

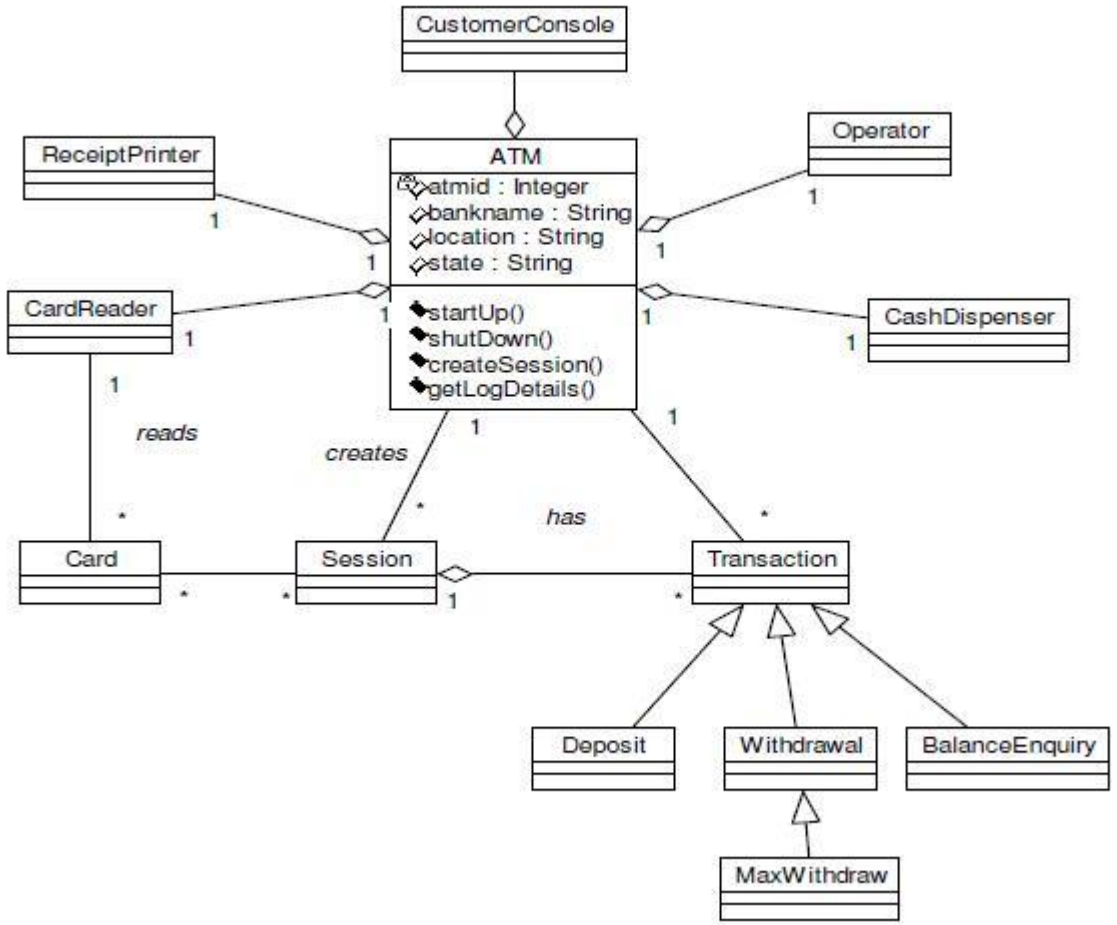
UNIFIED MODELLING LANGUAGE

CLASS DIAGRAM EXAMPLE– WASHING MACHINE

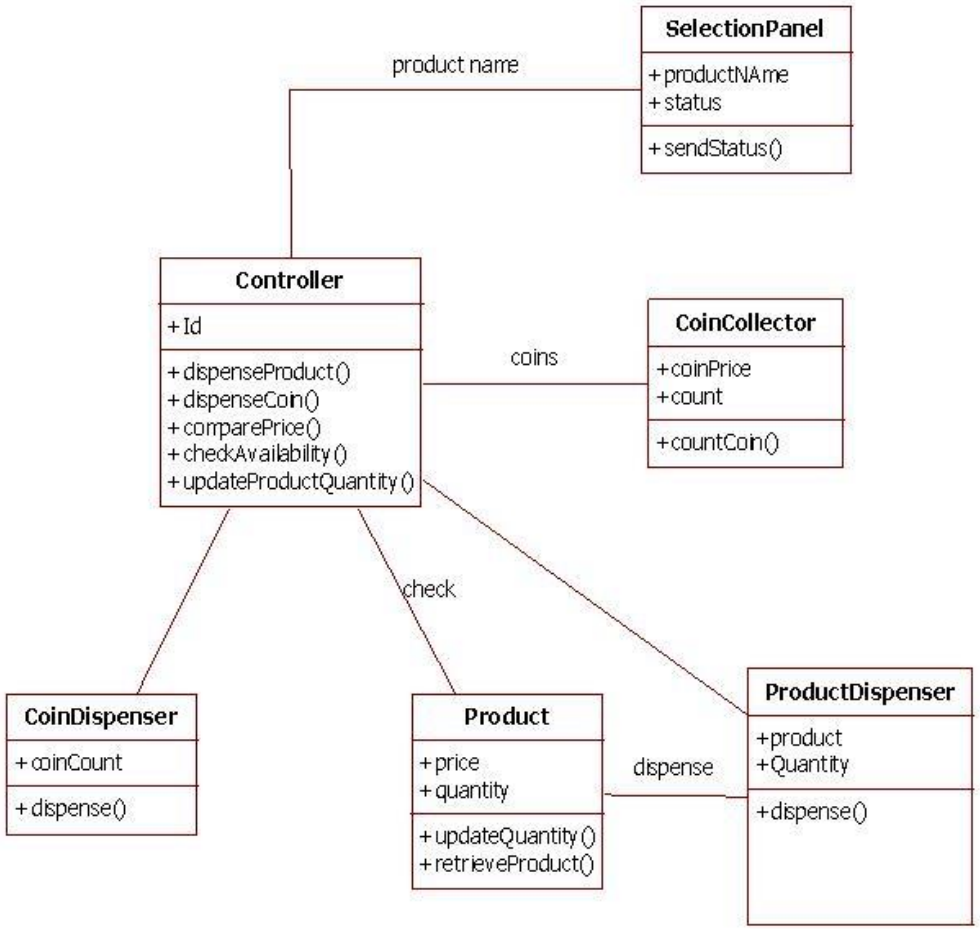


UNIFIED MODELLING LANGUAGE

CLASS DIAGRAM EXAMPLE– ATM MACHINE



CLASS DIAGRAM EXAMPLE – VENDING MACHINE



UNIFIED MODELLING LANGUAGE

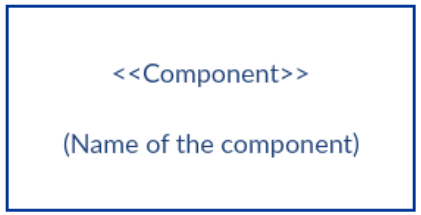
COMPONENT DIAGRAM

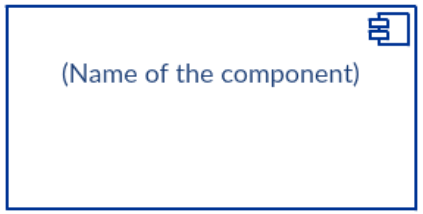
- A component diagram in UML for an embedded system typically illustrates the various components or modules within the system and their relationships.
- Component diagrams are used to model the static implementation view of a system.
- Component diagrams are essentially class diagrams that focus on a system's components.
- Components diagram helps to
  - ✓ Imagine the system's physical structure
  - ✓ Pay attention to system components and how they relate
  - ✓ Emphasize the service behaviour as it relate to the interface
- Component diagrams commonly contain Components, Interfaces and Dependency, generalization, association, and realization relationships.

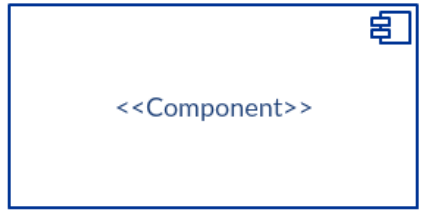
UNIFIED MODELLING LANGUAGE

COMPONENT DIAGRAM - NOTATIONS

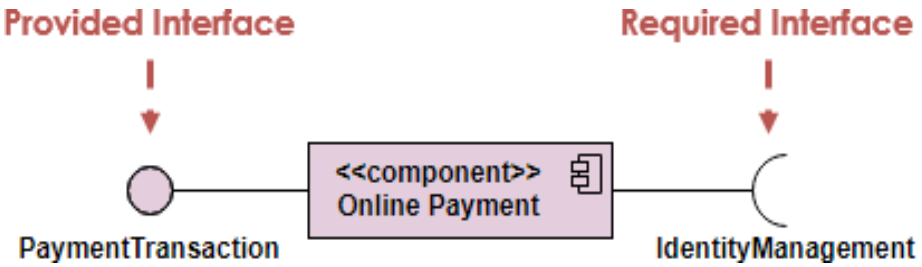
- Component – Logical unit block of the system. Three type of component symbol.





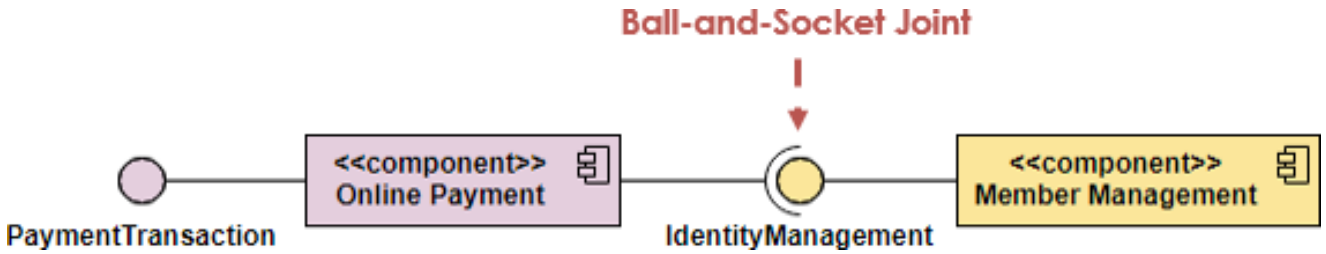


- Component Interfaces - show how components are wired together with each other.
  - Provide Interface: Define "a set of public attributes and operations that must be provided by the classes that implement a given interface".
  - Required Interface: Define "a set of public attributes and operations that are required by the classes that depend upon a given interface"

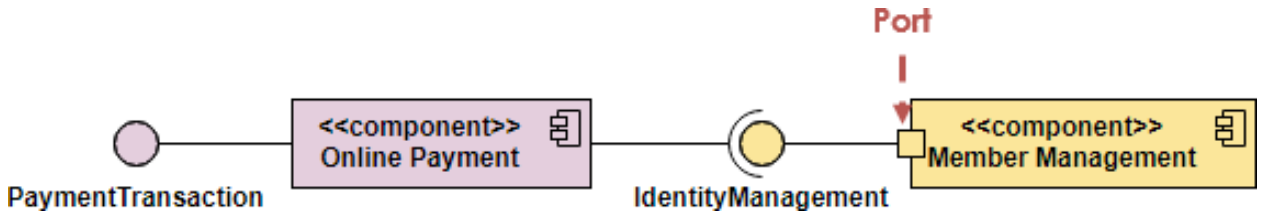


COMPONENT DIAGRAM - NOTATIONS

- Component Assemblies - Components can be "wired" together using to form subsystems, with the use of a ball-and-socket joint.



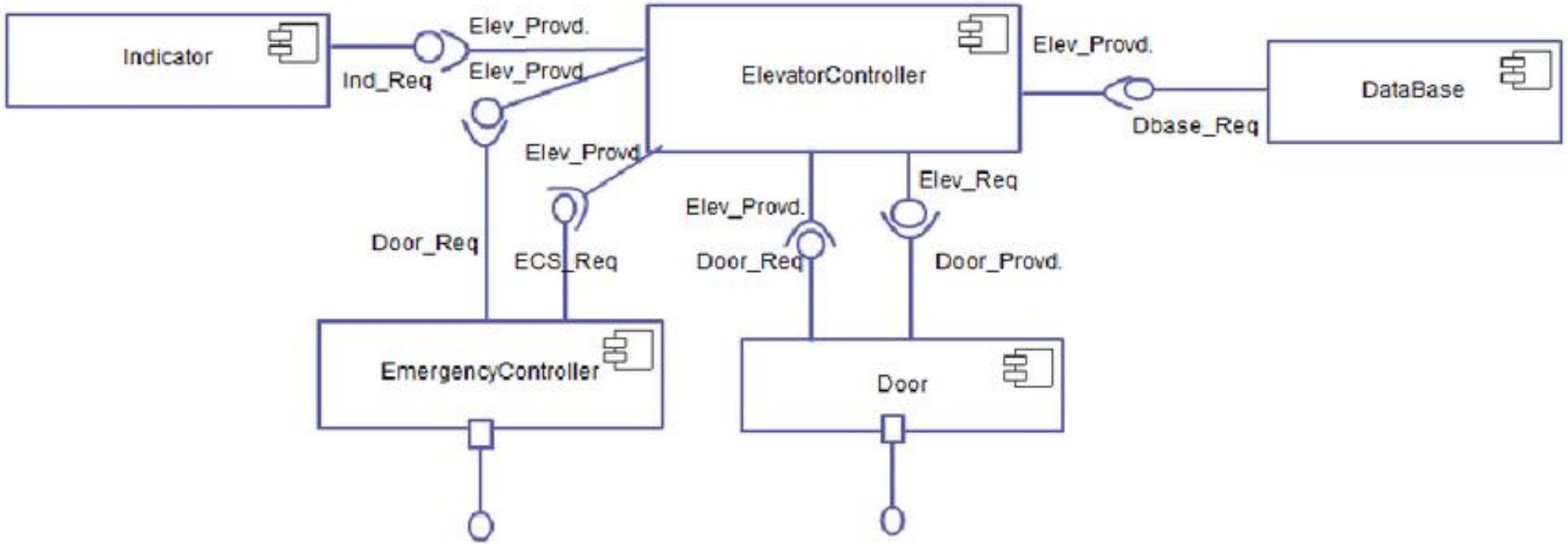
- Port : It indicates that the component itself does not provide the required interfaces



- Dependencies: draw dependencies among components using dotted lines

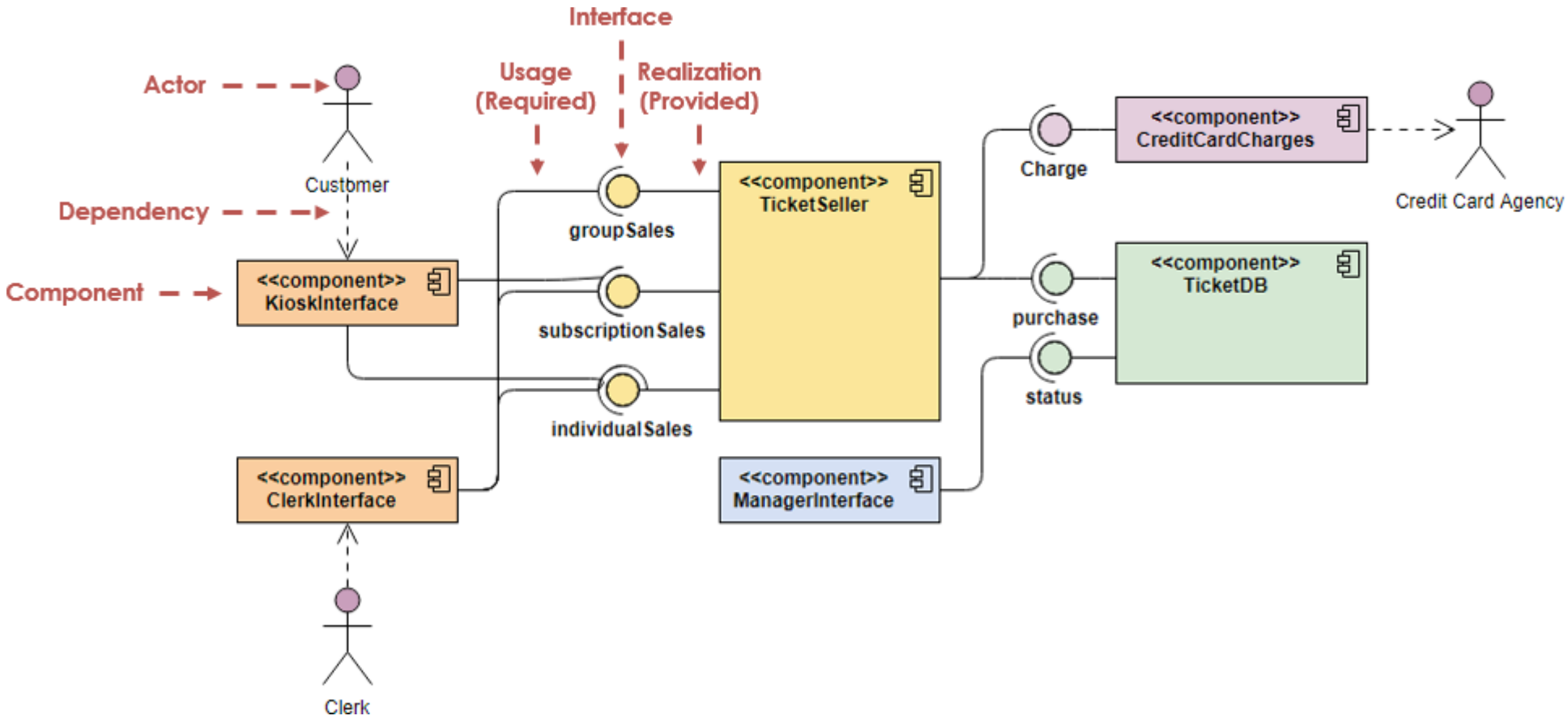
UNIFIED MODELLING LANGUAGE

COMPONENT DIAGRAM EXAMPLE – ELEVATOR SYSTEM



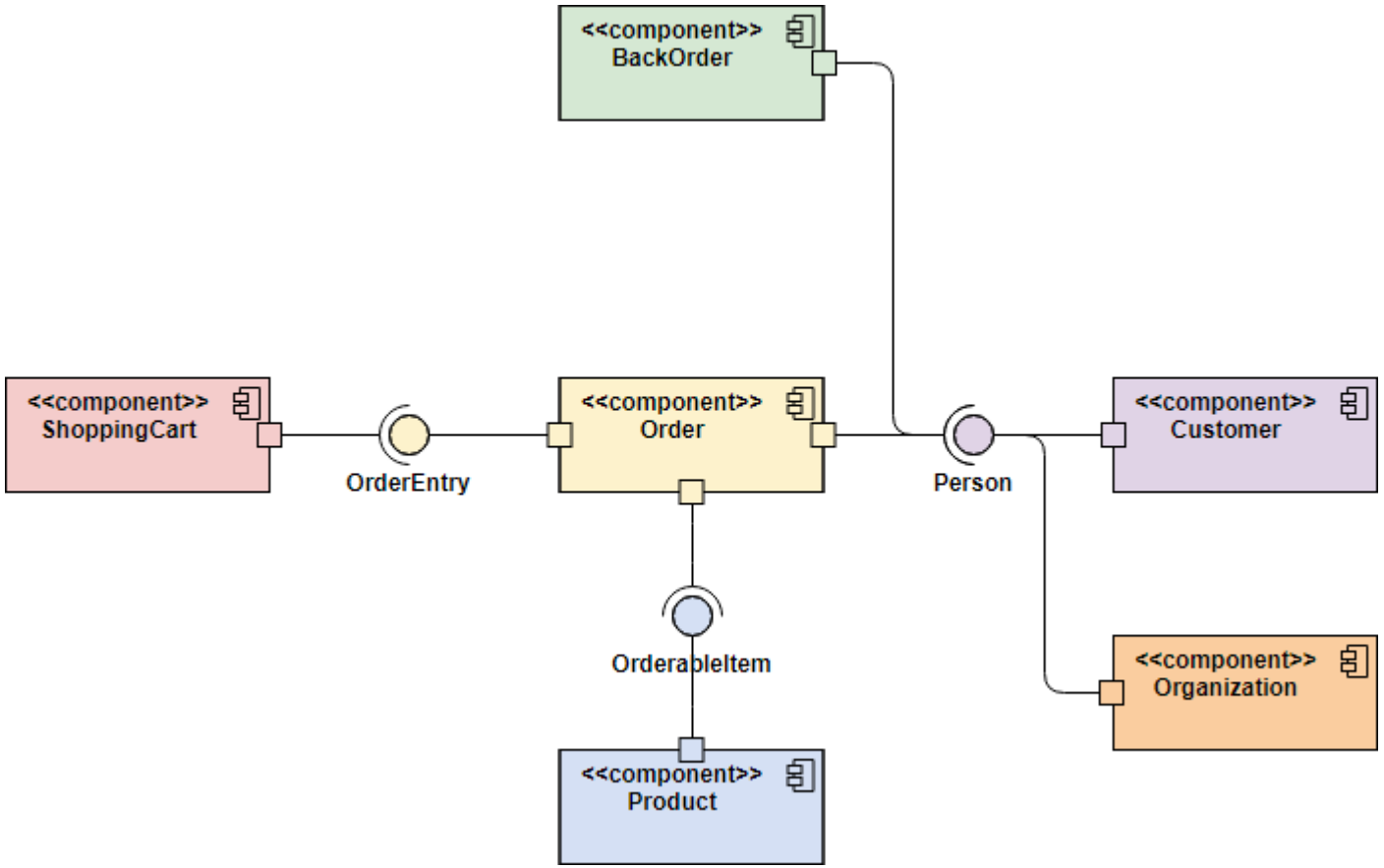
UNIFIED MODELLING LANGUAGE

COMPONENT DIAGRAM EXAMPLE - TICKET SELLING SYSTEM



UNIFIED MODELLING LANGUAGE

COMPONENT DIAGRAM EXAMPLE - ORDER MANAGEMENT SYSTEM



PETRINET MODEL

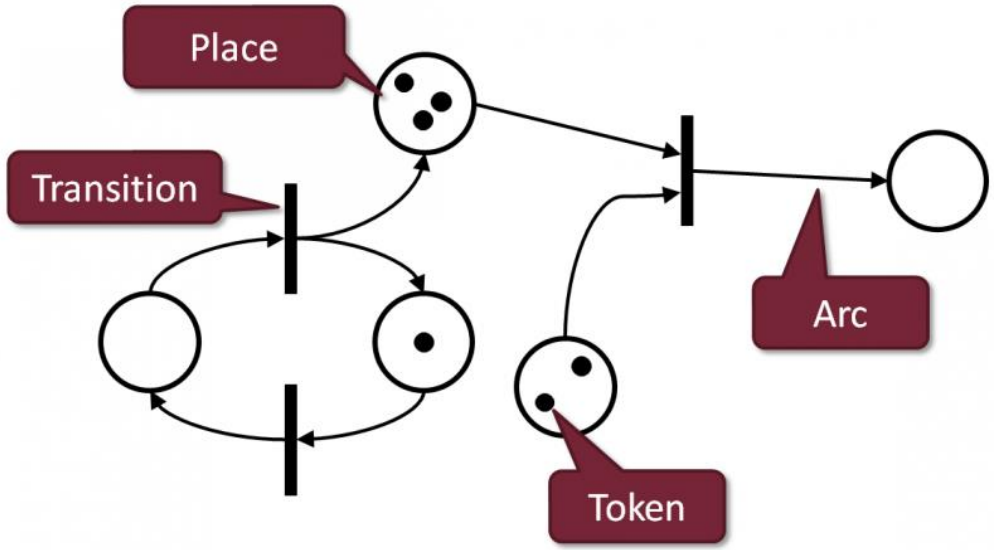
PETRINET MODEL

- Petri Nets (PN) are a graphical representation for the formal description of the logical interactions among parts or of the flow of activities in complex systems.
- Petri nets can be used to model a wide range of systems, including manufacturing processes, communication protocols, workflow systems, and more.
- It is mainly used for understanding the behaviour of complex systems and analysing their properties.
- Petri Net(PN) is very similar to State transition diagrams which model the system behaviour
- It is an abstract model to show the interaction between asynchronous processes
- In asynchronous process, start and sequence of processes may vary during the execution

PETRINET MODEL

COMPONENTS OF PETRINET MODEL

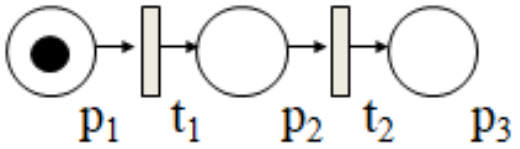
- Places represent conditions or states within the system (circles or ovals)
- Transitions represent events or actions that can occur in the system (bars or rectangles)
- Tokens are used to represent the state of the system and flow through the model (small solid dots)
- Arcs are directed connections between places and transitions, or transitions and places, indicating the flow of tokens (arrow)



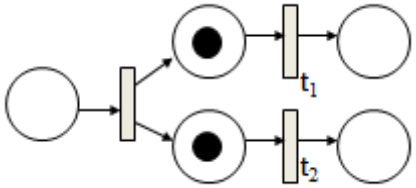
PETRINET MODEL

PROPERTIES OF PETRINET MODEL

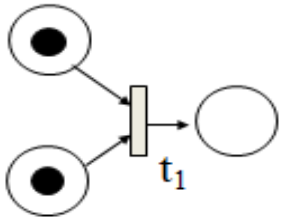
- Firing a transition refers to the occurrence of a event. Once firing is initiated, tokens will be transferred from input to output places
- Sequential Execution: Transition t2 can take place only after t1. Here a constraint is imposed "t2 after t1"



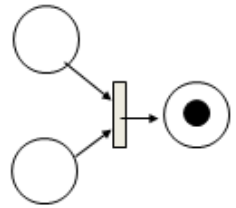
- Concurrency: This property is used in modelling distributed control system. t1 and t2 are concurrent process



- Merging: When several tokens are arrived at same transition merging will take place



- Synchronization: Transition will be enabled only when at least one token is available at each of its input places



PETRINET MODEL

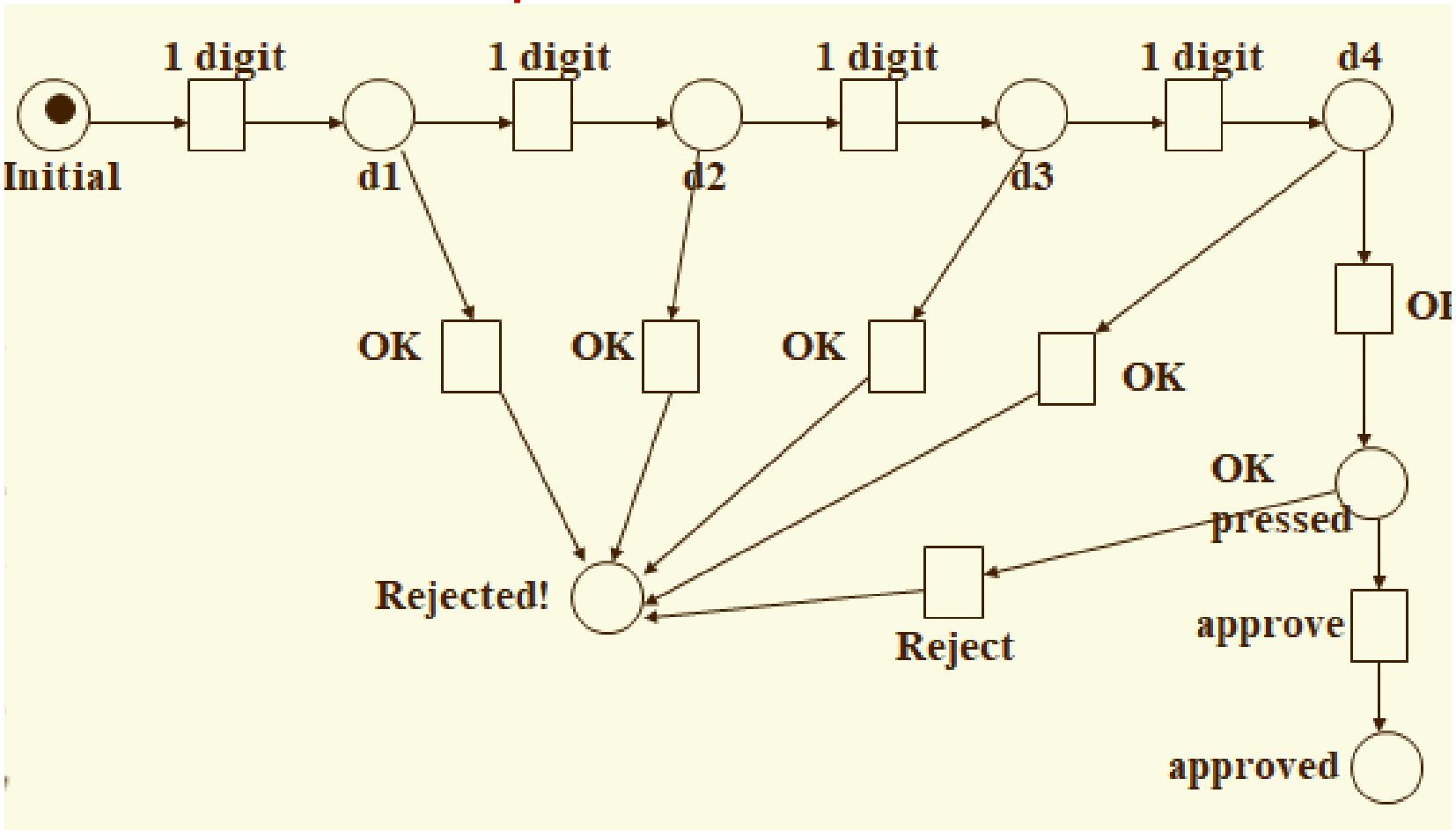
Example-1: Point of sale terminal

- A point of sale terminal (POS terminal) is an electronic device used to process card payments at retail locations.

- A POS terminal generally does the following:
  - ✓ Reads the information off a customer's credit or debit card
  - ✓ Validate the user using 4-digit PIN number
  - ✓ Checks whether the funds in a customer's bank account are sufficient
  - ✓ Transfers the funds from the customer's account to the seller's account
  - ✓ Records the transaction and prints a receipt

PETRINET MODEL

Example-1: Point of sale terminal



Scenario 1: Normal - Enters all 4 digits and press OK.

Scenario 2: Exceptional - Enters only 3 digits and press OK.

PETRINET MODEL

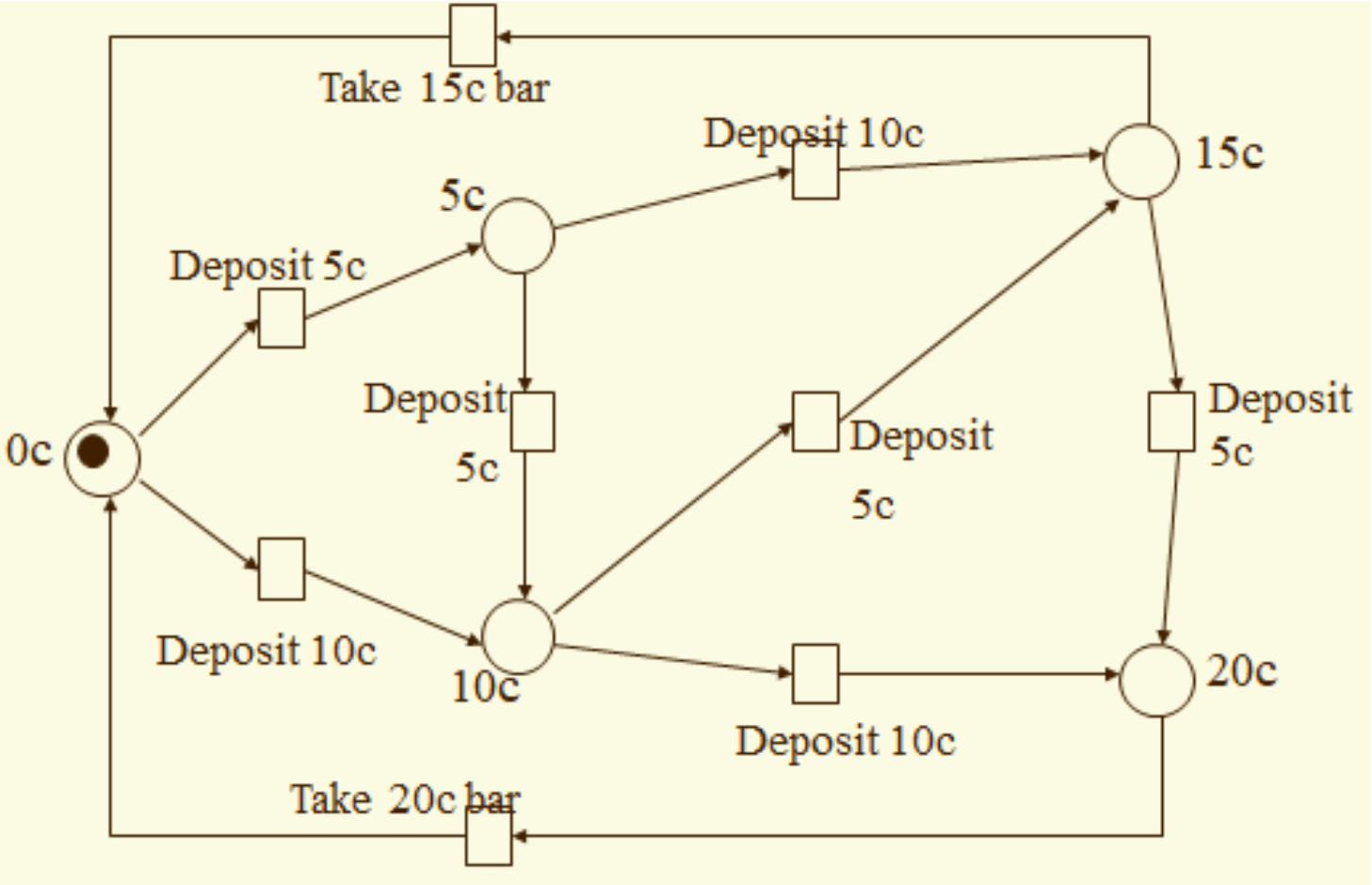
Example-2: VENDING MACHINE

- The machine dispenses two kinds of snack bars 20c and 15c. Constraint: 10c and 5c coins can only be used.

Scenario 1: Deposit four 5c, take 20c snack bar.

Scenario 2: Deposit 10 + 5c, take 15c snack bar.

Scenario 3: Deposit 5 + 10 + 5c, take 20c snack bar.



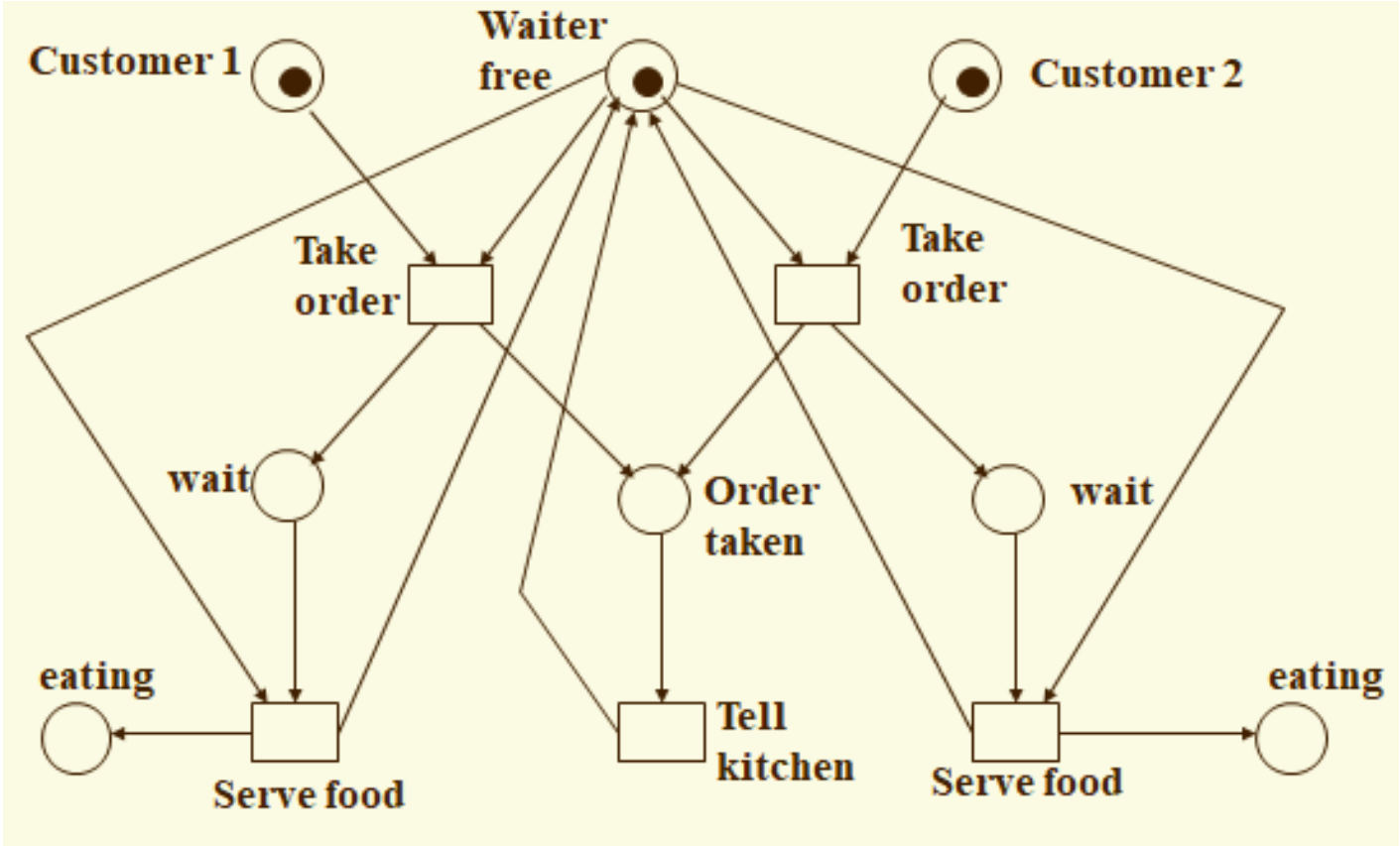
PETRINET MODEL

Example-3: ORDER MANAGEMENT IN A RESTAURANT

- In the real world, many events are concurrent in nature. Many applications have global state formed from local state.

- Scenario 1:
  - Waiter takes order from customer 1
  - Serves customer 1
  - Takes order from customer 2
  - Serves customer 2.

- Scenario 2:
  - Waiter takes order from customer 1
  - Takes order from customer 2
  - Serves customer 2
  - Serves customer 1



FINITE STATE MACHINE

FINITE STATE MACHINE

- Finite State Machines (FSM) is a conceptual model to represent the discrete interactions in a system. It represents how one single activity can change its behaviour over time, reaction to internally or externally triggered events.

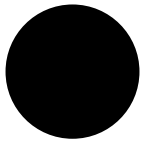
- There is finite number of possible states in a system and a system can only exist in one of these at an instance.

- There can be a transition of the present state to the next state, which depends on the inputs and state transition function.

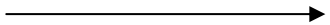
- Different types of state machines:
  - Mealy machine
  - Moore machine
  - Harel state charts
  - **UML state machines**

FINITE STATE MACHINE

NOTATIONS

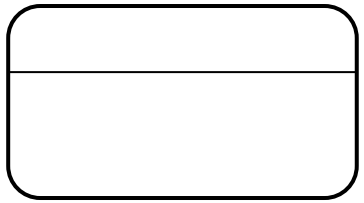


Initial state

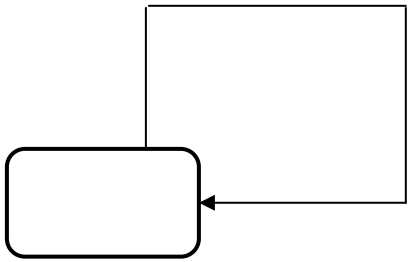


Transition Symbol

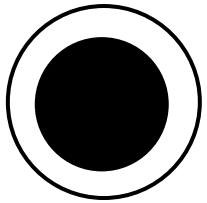
State



Composite State



Self Transition



Final state

FINITE STATE MACHINE

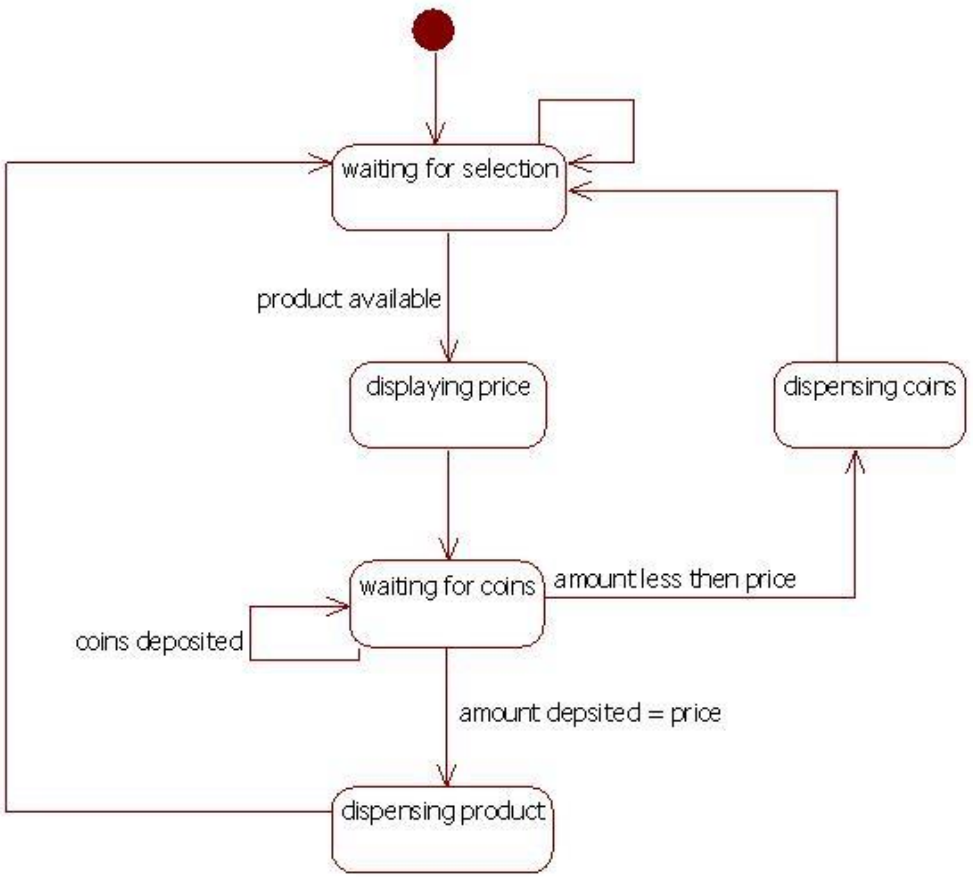
- FSM consists of a finite number of states, transitions between states, and actions.
  - State represents a certain behaviour.
  - Transition indicates a state change and is guarded by a condition.
  - Action is a description of an activity that is to be performed.

- Uses of FSM diagram:
  - ✓ We use it to depict event driven objects
  - ✓ We use it for showing use cases in business context
  - ✓ Shows the overall behaviour of state machine
  - ✓ We use it to model the dynamic behaviour of the system

- Steps to draw a state diagram:
  1. Identify the initial state and the final states.
  2. Identify the possible states in which the object can exist
  3. Label the events which trigger these transitions.

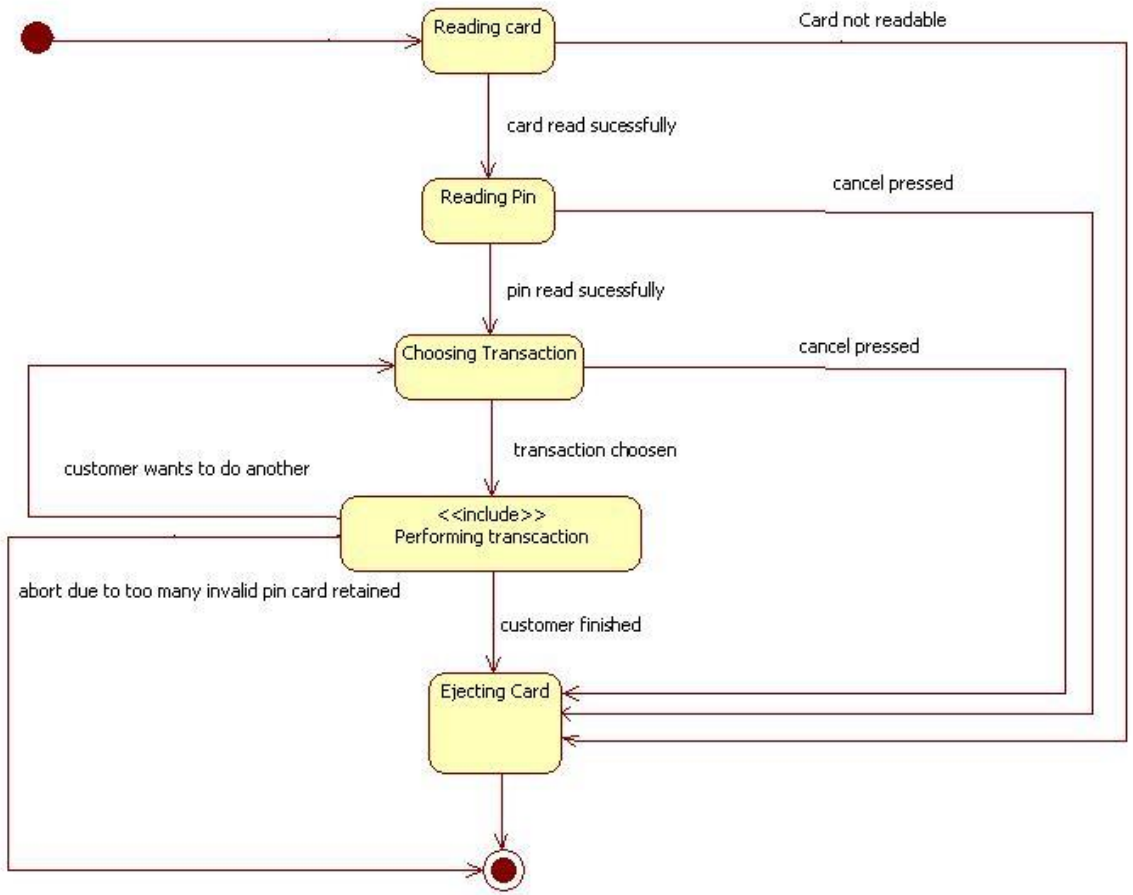
FINITE STATE MACHINE

Example-1: Simple Vending Machine



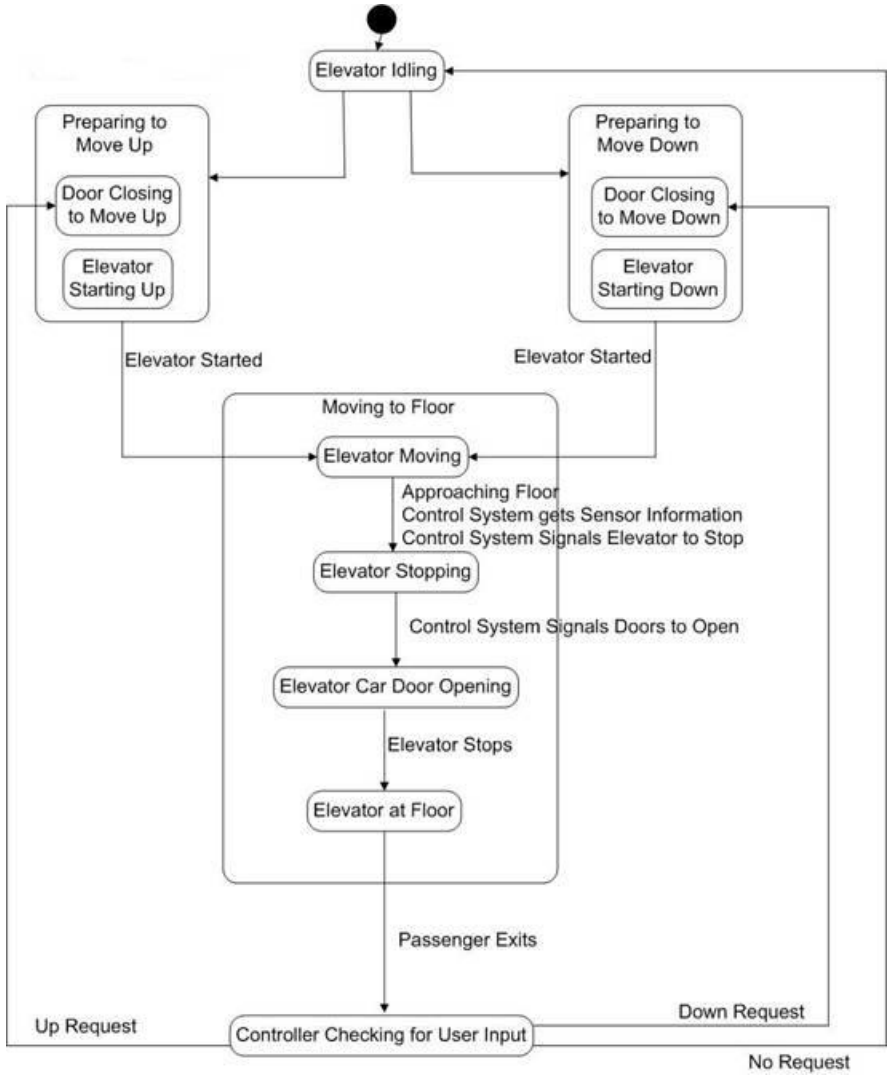
FINITE STATE MACHINE

Example-2: ATM Machine

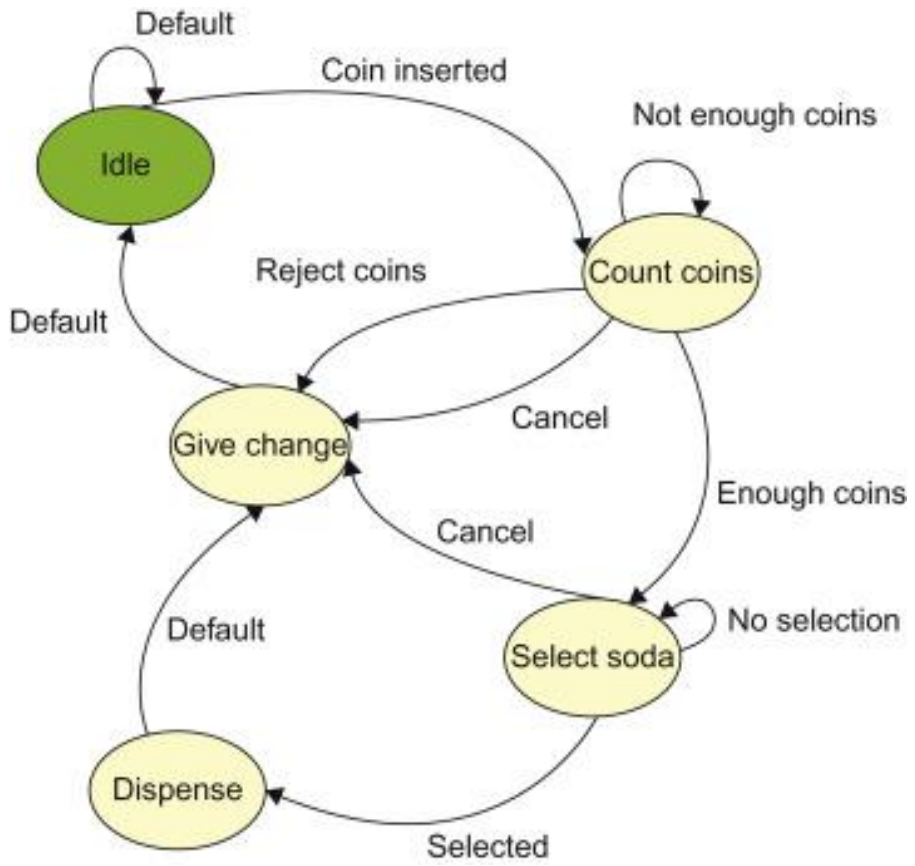


FINITE STATE MACHINE

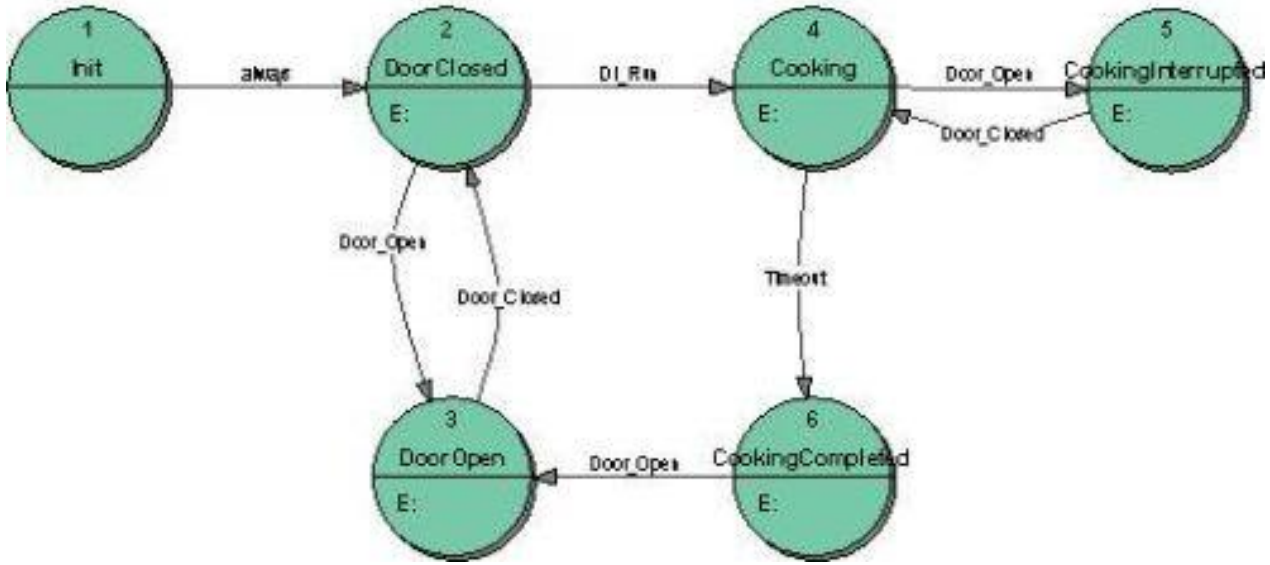
Example-3: Elevator Control System



FINITE STATE MACHINE



Example-1: Simple Soda Vending Machine



Example-2: Microwave oven

CONTROL DATA FLOW GRAPH

CONTROL DATA FLOW GRAPH

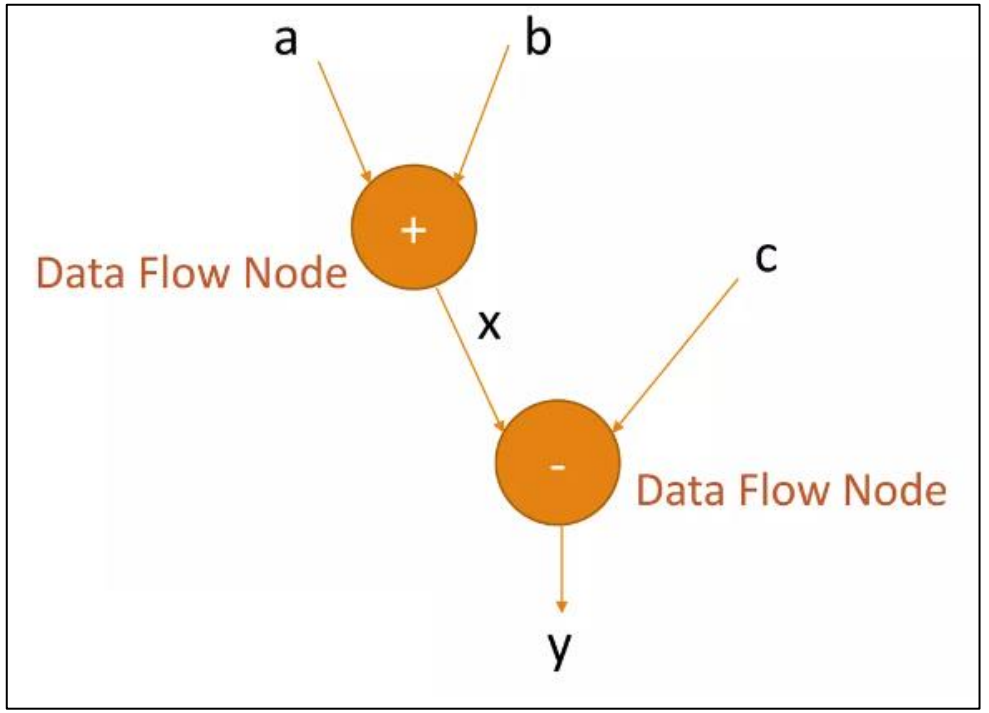
DATA FLOW GRAPH (DFG)

- The Data Flow Graph (DFG) model translates the data processing requirements into a data flow graph.
- It is a data driven model in which the program execution is determined by data.
- This model emphasises on the data and operations on the data which transforms the input data to output data.
- Embedded applications which are computational intensive and data driven are modelled using the DFG model. DSP applications are typical examples for it.
- Data Flow Graph (DFG) is a visual model in which the operation on the data (process) is represented using a block (circle) and data flow is represented using arrows.

CONTROL DATA FLOW GRAPH

DATA FLOW GRAPH (DFG)

- An inward arrow to the process (circle) represents input data and an outward arrow from the process (circle) represents output data in DFG notation.
- Suppose one of the functions in our application contains the computational requirement $x = a + b$, $y = x - c$
- Figure illustrates the implementation of a DFG model for implementing these requirements.



CONTROL DATA FLOW GRAPH

CONTROL DATA FLOW GRAPH (CDFG)

- The DFG (Data-Flow Graph) model is a data driven model in which the execution is controlled by data and it doesn't involve any control operations (conditionals).
- The Control DFG (CDFG) model is used for modelling applications involving conditional program execution.
- CDFG models contains both data operations and control operations.
- CDFG model graphically represents the conditions and the program flow along a condition dependent path.
- CDFG contains both data flow nodes and decision nodes, whereas DFG contains only data flow nodes.

CONTROL DATA FLOW GRAPH

CONTROL DATA FLOW GRAPH (CDFG)

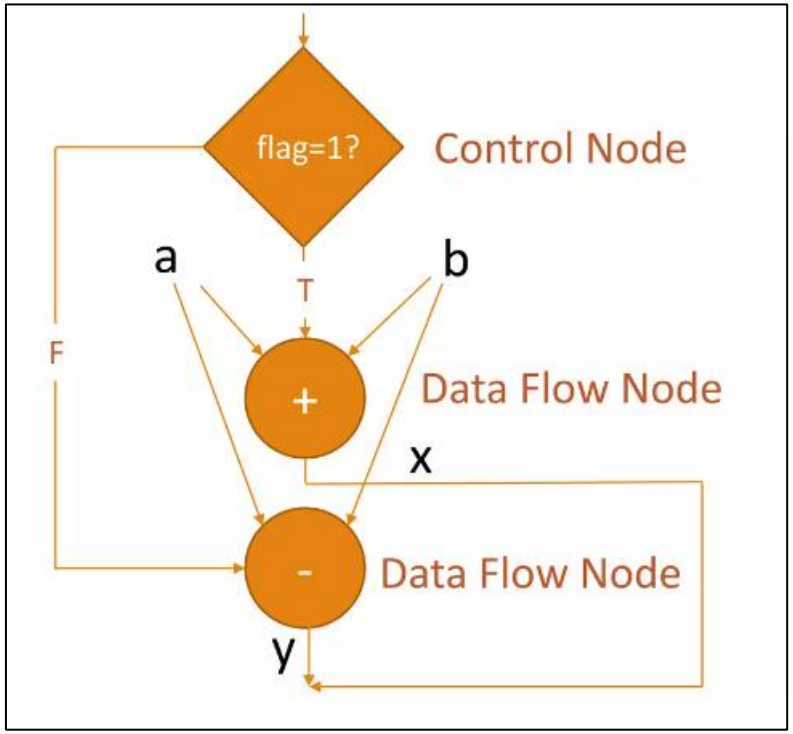
- Consider the implementation of the CDFG for the following requirement. If flag= 1 ,x=a+b; else y = a - b

- This requirement contains a decision making process. The CDFG model for the same is given in the figure.

- The control node is represented by a 'Diamond' block which is the decision making element.

- CDFG translates the requirement, which is modelled to a concurrent process model.

- The decision on which process is to be executed is determined by the control node.



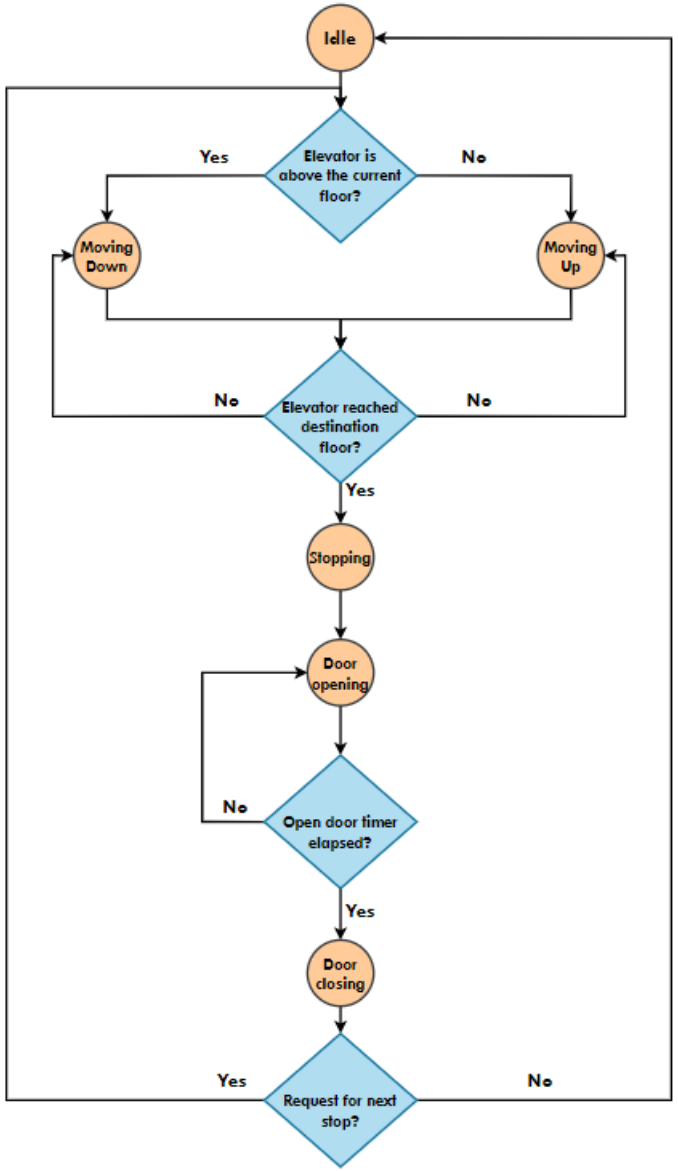
CONTROL DATA FLOW GRAPH

CONTROL DATA FLOW GRAPH (CDFG)

- A real world example for modelling the embedded application using CDFG is capturing and saving of the image to a format set by the user in a digital still camera.
- Here everything is data driven.
  - Analog Front End converts the CCD sensor generated analog signal to Digital Signal
  - The data from ADC is stored to a frame buffer for the use of a media processor which performs various operations like, auto correction, white balance adjusting, etc.
- The decision on, in which format the image is stored (formats like JPEG, TIFF, BMP, etc.) is controlled by the camera settings, configured by the user.

CONTROL DATA FLOW GRAPH

CONTROL DATA FLOW GRAPH (CDFG)
EXAMPLE – ELEVATOR SYSTEM



THANK YOU