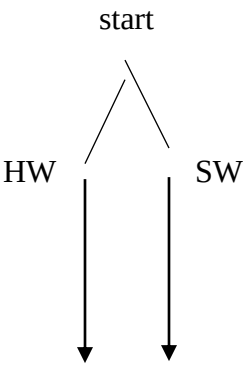
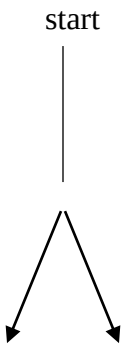
Hardware and Software Co-Design

Digital systems designs consists of hardware components and software programs that execute on the hardware platforms. Hardware and software co-design can be defined as meeting System level objectives by exploiting the cooperation between hardware and software through their concurrent design.

In Traditional Co-design, both hardware and software are designedby independent groups of experts. But in concurrent design, both hardware and software are designed by same group of experts with cooperation.





Fundamental Issues in Hardware Software Co-Design

The hardware and software co-design is problem statement and when we try to solve this problem statement in real world, we may come across multiple issues in the design.

Fundamental issues are as follows:

1) Selecting the model

In hardware software co-design, models are used for capturing and describing the system characteristics. A model is formal system consisting of objects and composition rules. It is hard to choose which model should be followed in particular design. Most often, designers switch between a varieties of model from requirement specification, as objective varies with each phase.

2) Selecting the architecture

The architecture specifies how a system is going to implement in terms of number of types of different components and the interconnection between them. Common classification of architecture are as follows:

a) Application Specific Architecture Class (Controller Architecture)

b) General purpose class (CISC, RISC)

c) Parallel processing class(VLIW,SIMD, MIMD)

The **Controller Architecture** implements the finite state machine model using a state register and the combinational circuits. The state register holds the present state and the combinational circuits implements the logic for next state and output.

The **Datapath Architecture** implements the data flow graph model where the output is generated as a result of a set of predefined computations on the input data. A datapath represents a channel between the input and output. In datapath Architecture, datapath may contain registers, counters, register files, memories and ports along with high speed arithmetic unit.

The **Finite State Machine Datapath** (FSMD) architecture combines the controller architecture with datapatharchitecture. It implements a controllerwith datapath. The controller generates the control inputs whereas the datapath process the data. The datapath has two type of I/O ports, one acts as a control port for receiving/ sending the control inputs and second one interfaces the datapath with external world for data input and output.

The **Complex Instruction Set Computing** (CISC) architecture uses an instruction set representing complex operations. It is possible for a CISC instruction set to perform a large complex operation with a single instruction. The use of single instruction greatly reduces the program memory access and program memory size requirement.

The **Reduced Instruction Set Computing** (RISC) uses instruction set representing simple operations and it requires the execution of multiple RISC instruction to perform complex instruction. The datapath of RISC architecture contain large register file for storing operand and output. It supports extensive pipeling.

The Very Long Instruction Word (VLIW) architecture implements multiple functional units in the datapath. The VLIW instruction packages one standard instruction per functional units of the datapath.

Parallel processing architecture implements multiple concurrent processing elements (PE) and each processing element may associate a datapath containing register and local memory.

In **Single Instruction Multiple Data (SIMD),** a single instruction is executed in parallel with the help of the processing element (PE). The scheduling of instruction execution and controlling of each PE is performed by single controller.

In **Multiple Instruction Multiple Data** (MIMD), the processing elements executes different instruction at a given point of time. The MIMD architecture forms the basis of multiprocessing systems. The PE's in the multiprocessing systems communicates through mechanisms like shared memory and message passing.

3) Selecting the language

A programming language captures a Computational model and maps it into architecture. A model can captured using multiple programming languages like C, C++, C#, Java etc. for software implementation and languages like VHDL, System C, Verilog etc. for hardware implementations. C++ is a good candidate for capturing an object oriented model.

4) Partitioning System Requirements into hardware and software

From an implementing perspective, it may be possible to implement the system requirements in either hardware or software. It is a tough decision making task to figure out which one to opt.Various hardware and software trade-offs are used for making a decision on the hardware-software partitioning.

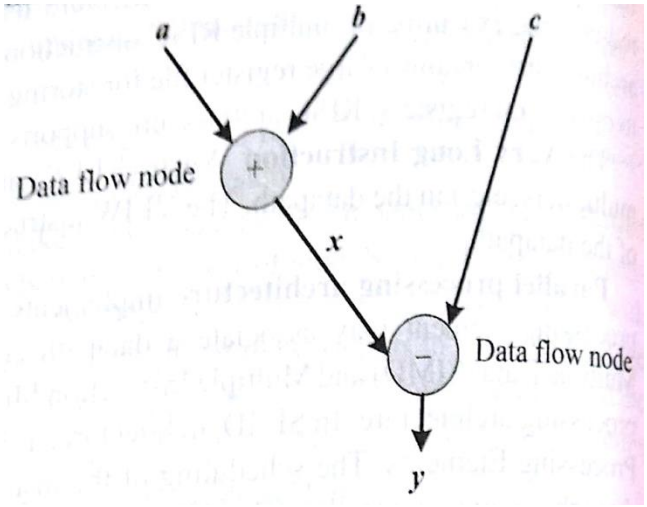
Computational Models

Data Flow Graph (DFG) model, State Machine model, Concurrent Process model, Sequence Program Model, Object Oriented Model etc. are commonly used computational model.

1. Data Flow Graph

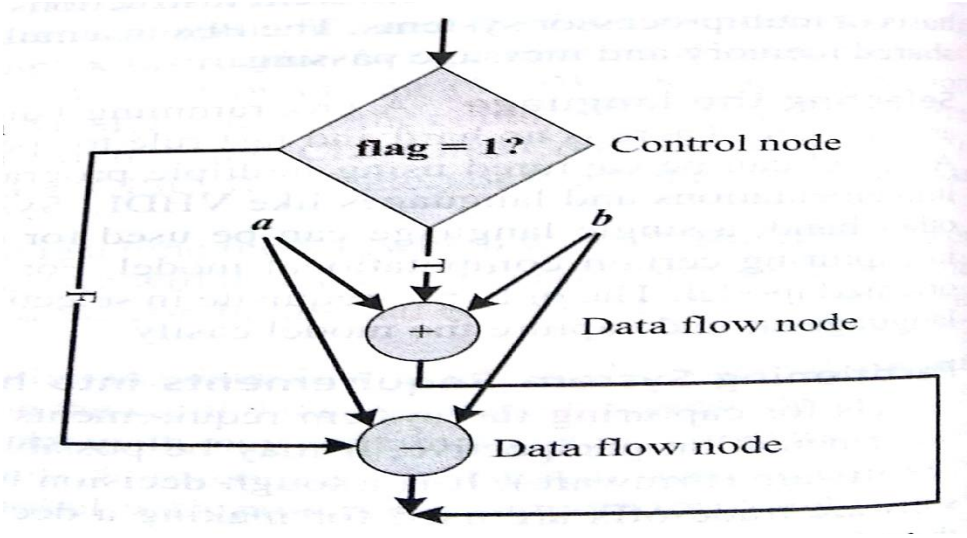
- Translates the data processing requirement into data flow graph.

- Data driven model in which program execution is determined by data.
- Emphasizes on data and operation on data which transforms the input data to output.
- Translates the program as a single sequential process execution.
- Visual model in which operation on data is represented in circle and data flow is represented by directed arrow.
- An inward arrow to the circle represents the input data and an outward arrow from the circle represents the output data in DFG.
- A datapath is the data flow path from input to output.
- A DFG model is said to be acyclic DFG, if it doesn't have multiple values for input variables and output variables.



2. Control Data Flow Graph

- The control data flow graph model is used for modelling application involving conditional program execution.
- CDFG models contains both data operations and control operations.
- CDFG uses Data Flow Graphs as element and conditional as decision makers.
- CDFG contain both data flow nodes and decision nodes.
- Control node is represented by diamond block which is the decisionmaking element in normal flow chart design.
- CDFG translates the requirement to a concurrent process model.



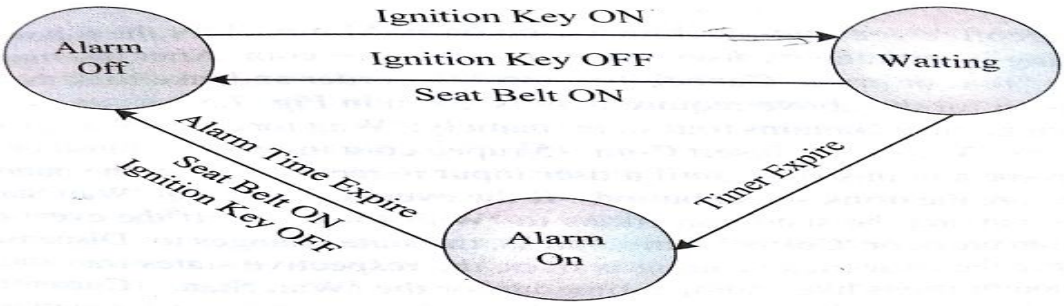
3. State Machine Model

The state machine is used for modelling reactive or event-driven system for processing behavior are dependent on state transitions. The state machine describes the system behavior with States, Events, Actions and Transitions. State is a representation of a current situation. An event is an input to state. The event act as stimuli for state transition. Transition is the movement from one state to another. Action is the activity to be performed by the state machine.

Seat Belt Warning in an automotive using FSM model is as explained. The system requirement are captured as

- When the ignition is turned on and the seat beat is not fastened within 10 seconds of ignition ON, the system generates an alarm signal for 5 seconds.
- The Alarm is turned off when the alarm time expires after 5 seconds or if the driver fastens the belt or if the ignition switch is turned off, whichever happens first.

Here states are Alarm Off, Waiting and Alarm On ad the events are ignition key ON, ignition key OFF, Timer Expire, Alarm Time Expire and Seat Belt ON. FSM is as shown below.



Sequential Program Model

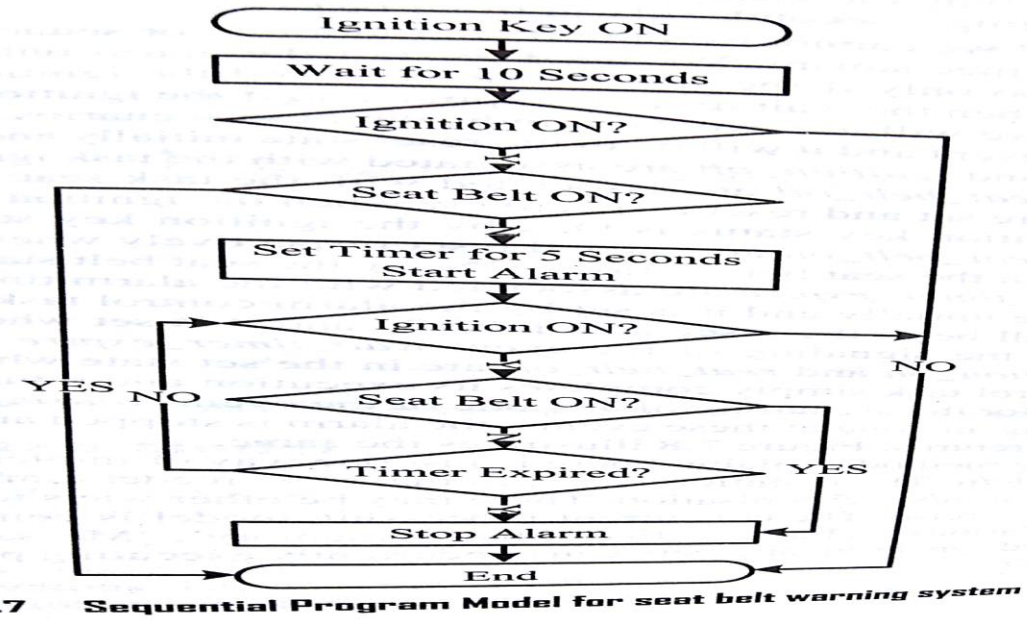
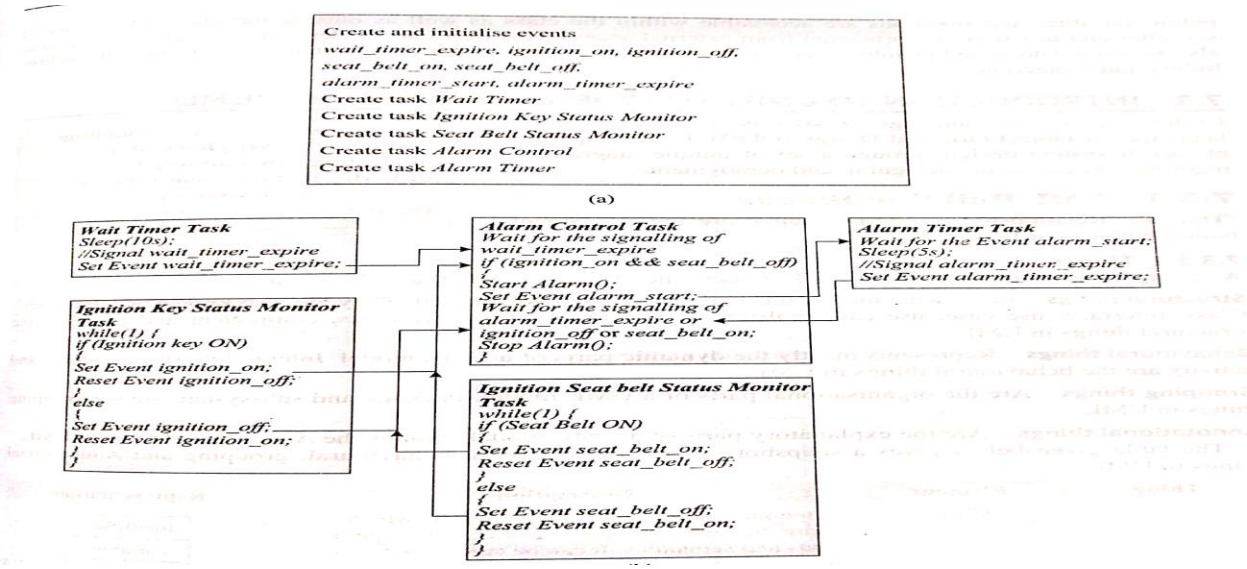


Fig. 7.7 Sequential Program Model for seat belt warning system

In the sequential programming model, the functions or processing requirements are executed in sequence. It is same as the conventional procedural programming. Here the program instruction are iterated and executed conditionally and the data gets transformed through a series of operations. FSM's are good choice for sequential program modelling. Another important tool used for modelling sequential program is Flow charts. The FSM represents states, events, transitions and actions, whereas Flow charts models the execution flow. Figure illustrate the flow chart approach of for modelling the Seat Belt Warning system.

Concurrent/Communicating Process Model

The concurrent or communicating process model models concurrently executing task/process. It is easier to implement certain requirements in concurrent processing model than conventional sequential execution. Sequential execution leads to a single sequential execution of task and thereby leads to poor processor utilization, when the task involves I/O waiting, sleeping for specific duration etc. If the task is split into multiple subtasks, it is possible to tackle the CPU usage effectively. Concurrent processing model requires additional overheads in task scheduling, task synchronization and communication.



Seat Belt Warning system can be designed using concurrent processing model with following 5 tasks:

1. Timer task for waiting 10 seconds(wait timer task)
2. Task for checking the ignition key status (ignition key status monitoring task)
3. Task for checking the seat belt status (seat belt status monitoring task)
4. Task for starting and stopping the alarm (alarm control task)
5. Alarm timer task for waiting 5 seconds (alarm timer tasks)

We have five tasks here and we cannot execute them randomly or sequentially. We need to synchronize their execution through some mechanism. For example, we need to start the alarm only after the expiration of the 10 seconds wait timer and that too if the seat belt is OFF and the ignition key is ON. Figure illustrates the concurrent processing program model of Seat Belt Warning System.

Object Oriented Model

The object oriented model is an object based model for modelling system requirements.It distributes a complex software requirement into simple well defined pieces called objects. In object oriented modelling, object is an entity used for representing or modelling a particular piece of the system. Each object is characterized by a set of unique behavior and state. A class is an abstract description of a set of objects and it can be considered as a blueprint of an object. A

class represents the state of an object through member variable and object behavior through member function. Member variable and function can be private, public or protected.

UML

Unified Modeling Language (UML) is a general purpose modelling language. The main aim of UML is define a standard way to visualize the way a system has been designed. It is quite similar to blueprints used in other fields of engineering.

Things

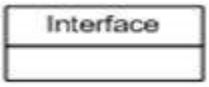
Things are the most important building blocks of UML. Things can be −

- Structural
- Behavioral
- Grouping
- Annotational

Structural things define the static part of the model. They represent the physical and conceptual elements. Following are the brief descriptions of the structural things.

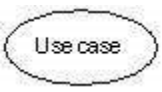
Class − Class represents a set of objects having similar responsibilities.

Interface − Interface defines a set of operations, which specify the responsibility of a class.

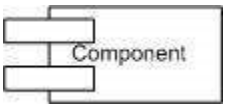


Collaboration −Collaboration defines an interaction between elements.

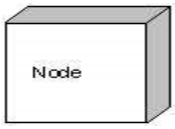
Use case −Use case represents a set of actions performed by a system for a specific goal.



Component −Component describes the physical part of a system.

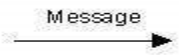


Node − A node can be defined as a physical element that exists at run time.



A behavioral thing consists of the dynamic parts of UML models. Following are the behavioral things −

Interaction − Interaction is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task.

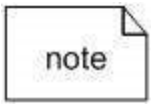


State machine − State machine is useful when the state of an object in its life cycle is important. It defines the sequence of states an object goes through in response to events. Events are external factors responsible for state change

Grouping things can be defined as a mechanism to group elements of a UML model together. There is only one grouping thing available −

Package − Package is the only one grouping thing available for gathering structural and behavioral things.

Annotational things can be defined as a mechanism to capture remarks, descriptions, and comments of UML model elements. **Note** - It is the only one Annotational thing available. A note is used to render comments, constraints, etc. of an UML element.

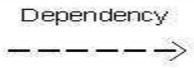


Relationship

Relationship is another most important building block of UML. It shows how the elements are associated with each other and this association describes the functionality of an application.

There are four kinds of relationships available.

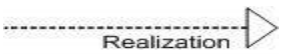
Dependency is a relationship between two things in which change in one element also affects the other.



Association is basically a set of links that connects the elements of a UML model. It also describes how many objects are taking part in that relationship.

Generalization can be defined as a relationship which connects a specialized element with a generalized element. It basically describes the inheritance relationship in the world of objects.

Realization can be defined as a relationship in which two elements are connected. One element describes some responsibility, which is not implemented and the other one implements them. This relationship exists in case of interfaces.



UML Diagram

UML is linked with object oriented design and analysis. UML makes the use of elements and forms associations between them to form diagrams. Diagrams in UML can be broadly classified as:

1. Structural Diagrams – Capture static aspects or structure of a system. Structural Diagrams include: Component Diagrams, Object Diagrams, Class Diagrams and Deployment Diagrams.
2. Behavior Diagrams – Capture dynamic aspects or behavior of the system. Behavior diagrams include: Use Case Diagrams, State Diagrams, Activity Diagrams and Interaction Diagrams.

Structural UML Diagrams

1. Class Diagram – The most widely use UML diagram is the class diagram. It is the building block of all object oriented software systems. We use class diagrams to depict the static structure of a system by showing system's classes,their methods and attributes. Class diagrams also help us identify relationship between different classes or objects.
2. Object Diagram – An Object Diagram can be referred to as a screenshot of the instances in a system and the relationship that exists between them. Since object diagrams depict behaviour when objects have been instantiated, we are able to study the behaviour of the system at a particular instant. An object diagram is similar to a class diagram except it shows the instances of classes in the system. We depict actual classifiers and their relationships making the use of class diagrams. On the other hand, an Object Diagram represents specific instances of classes and relationships between them at a point of time.
3. Component Diagram – Component diagrams are used to represent the how the physical components in a system have been organized. We use them for modelling implementation details. Component Diagrams depict the structural relationship between software system elements and help us in understanding if functional requirements have been covered by planned development. Component Diagrams become essential to use when we design and build complex systems. Interfaces are used by components of the system to communicate with each other.
4. Deployment Diagram – Deployment Diagrams are used to represent system hardware and its software.It tells us what hardware components exist and what software components run on them.We illustrate system architecture as distribution of software artifacts over

distributed targets. An artifact is the information that is generated by system software. They are primarily used when a software is being used, distributed or deployed over multiple machines with different configurations.

5. Package Diagram – We use Package Diagrams to depict how packages and their elements have been organized. A package diagram simply shows us the dependencies between different packages and internal composition of packages. Packages help us to organise UML diagrams into meaningful groups and make the diagram easy to understand. They are primarily used to organise class and use case diagrams.

Behavior Diagrams

1. State Machine Diagrams – A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioral diagram and it represents the behavior using finite state transitions. State diagrams are also referred to as State machines and State-chart Diagrams. These terms are often used interchangeably.So simply, a state diagram is used to model the dynamic behavior of a class in response to time and changing external stimuli.
2. Activity Diagrams – We use Activity Diagrams to illustrate the flow of control in a system. We can also use an activity diagram to refer to the steps involved in the execution of a use case. We model sequential and concurrent activities using activity diagrams. So, we basically depict workflows visually using an activity diagram.An activity diagram focuses on condition of flow and the sequence in which it happens. We describe or depict what causes a particular event using an activity diagram.
3. Use Case Diagrams – Use Case Diagrams are used to depict the functionality of a system or a part of a system. They are widely used to illustrate the functional requirements of the system and its interaction with external agents(actors). A use case is basically a diagram representing different scenarios where the system can be used. A use case diagram gives us a high level view of what the system or a part of the system does without going into implementation details.
4. Sequence Diagram – A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.We can also use the

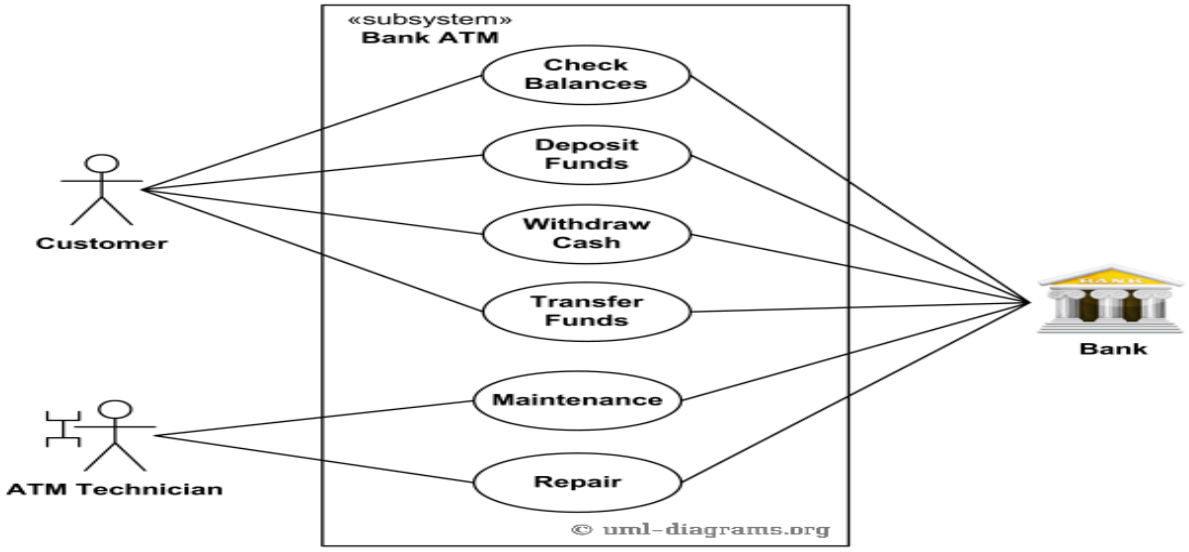
terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function. These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.

5. Communication Diagram – A Communication Diagram(known as Collaboration Diagram in UML 1.x) is used to show sequenced messages exchanged between objects. A communication diagram focuses primarily on objects and their relationships. We can represent similar information using Sequence diagrams,however, communication diagrams represent objects and links in a free form.

Case Study: ATM

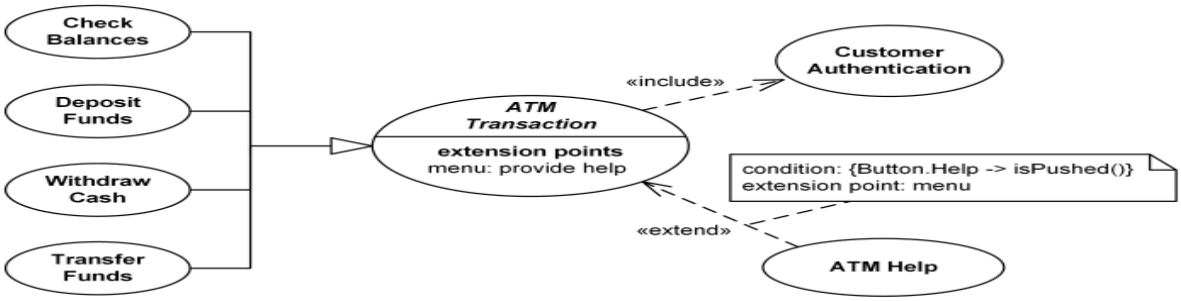
Use case diagram of ATM

An automated teller machine (ATM) or the automatic banking machine (ABM) is a banking subsystem (subject) that provides bank customers with access to financial transactions in a public space without the need for a cashier, clerk, or bank teller.Customer (actor) uses bank ATM to Check Balances of his/her bank accounts, Deposit Funds, Withdraw Cash and/or Transfer Funds (use cases). ATM Technician provides Maintenance and Repairs. All these use cases also involve Bank actor whether it is related to customer transactions or to the ATM servicing.

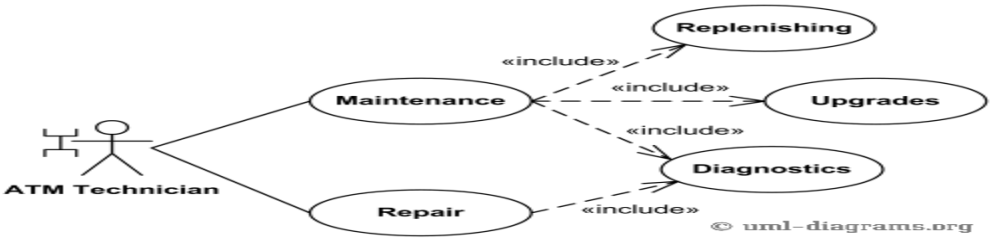


On most bank ATMs, the customer is authenticated by inserting a plastic ATM card and entering a personal identification number (PIN). *Customer Authentication* use case is required for every

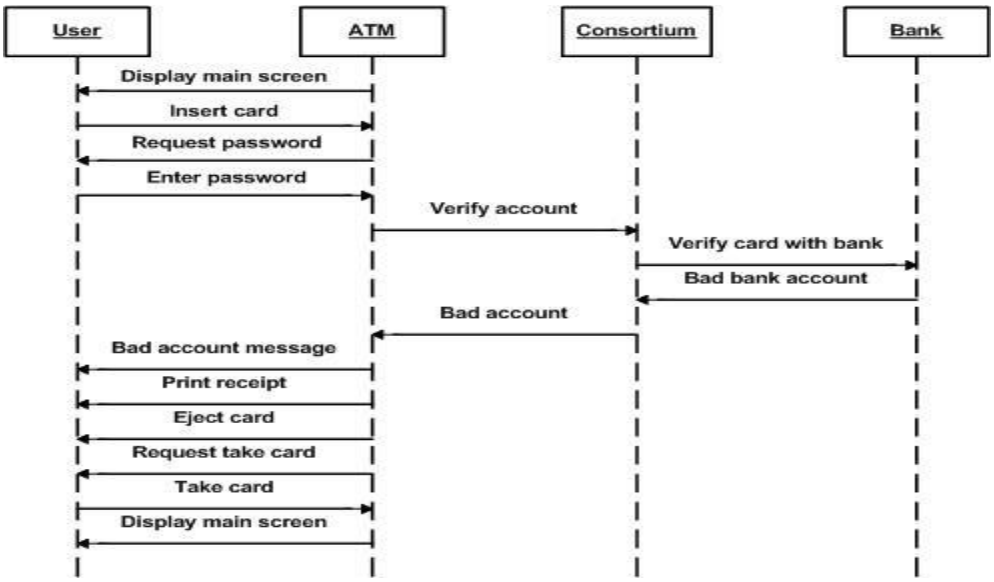
ATM transaction so we show it as **include** relationship. Including this use case as well as transaction**generalizations** make the *ATM Transaction* an **abstract use case**.



Customer may need some help from the ATM. *ATM Transaction* use case is extended via extension point called *menu* by the *ATM Help* use case whenever *ATM Transaction* is at the location specified by the *menu* and the bank customer requests help, e.g. by selecting Help menu item.

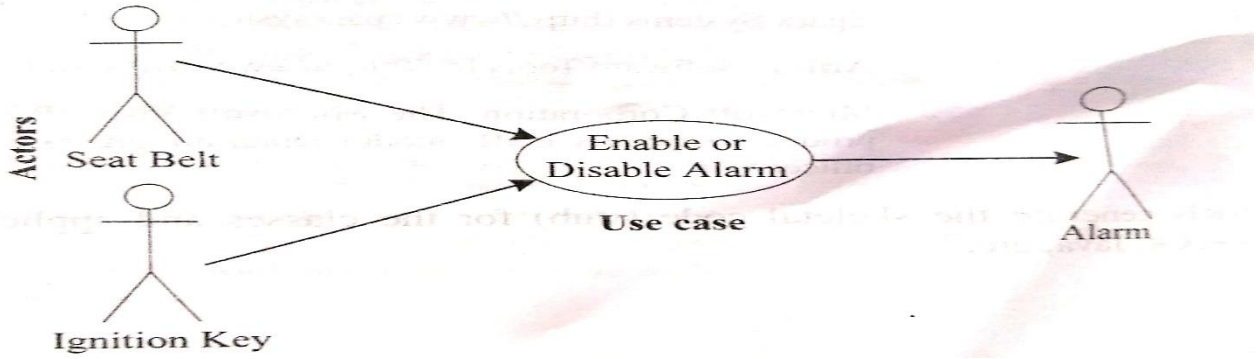


Sequence Diagram of ATM



Case Study: Seat belt

Use case diagram



Sequence Diagram