

Batch: A2 Roll No.: 16010322014
Experiment / assignment / tutorial No. 03
Grade: AA / AB / BB / BC / CC / CD /DD
Signature of the Staff In-charge with date

TITLE: System Design Analysis using UML

AIM: Design an Embedded System Model using UML/FSM Simulator

1. **OUTCOME:** Understand the concepts of co-design and co-simulation and testing methodology for Embedded system.

Simulator Used: -

- <https://app.diagrams.net/>
- [Draw.io \(diagrams.net\)](https://draw.io)

Hardware Architecture: -

- **Processor:** A microcontroller would serve as the "heart" of the ATM, running all its software and coordinating the hardware.
- **Memory:** This includes Program Memory to hold the ATM's operating software and Data Memory for transaction details.
- **Input Devices Interfacing Circuits:** These circuits would connect the processor to the ATM's keypad and card reader.
- **Outputs Interfacing Circuits:** These would drive the ATM's LCD display, receipt printer, and the mechanical cash dispenser.
- **Communication Ports:** A serial or network port would be used to connect the ATM to the bank's central servers.
- **Interrupt Controller and Timers:** These manage hardware requests and ensure operations happen within specific time periods, which is critical for a real-time system like an ATM

Software Architecture: -

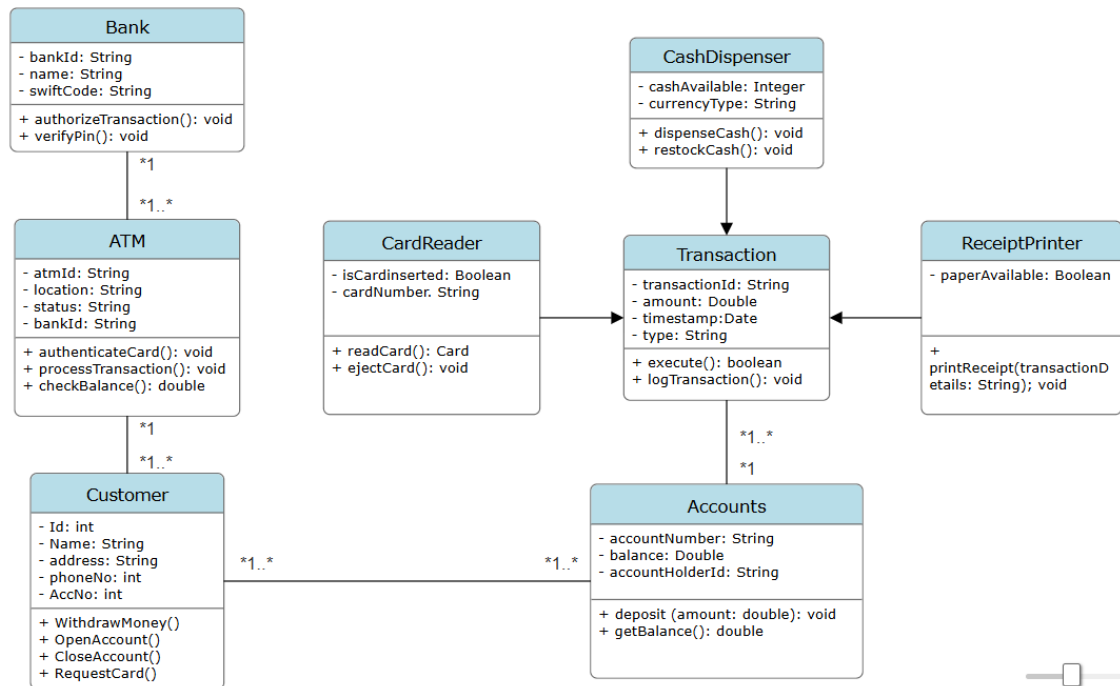
- **Application Software:** This is the top layer and would be the main program that the user interacts with. It includes the logic for `processTransaction()`, `checkBalance()`, and displaying menus on the screen.
- **API (Application Programming Interface):** This acts as the middleman between your ATM application and the Operating System. For example, when your application needs to print a receipt, it makes a call to the OS through the API.
- **Operating System:** For an ATM, this would be a Real-Time Operating System (RTOS) because it must respond to events within strict deadlines. This layer is made of several components:
 - **Kernel:** The core of the RTOS. It manages all the different software tasks (e.g., one task for the keypad, another for the dispenser), schedules them, and handles memory.
 - **Device Manager:** This uses specific device drivers to control the ATM's physical hardware, like the `CashDispenser` and `ReceiptPrinter`.

Department of Electronics and Telecommunication Engineering

Somaiya Vidyavihar University
K J Somaiya School of Engineering

- Communication Software: This manages the secure network connection to the bank for authorizing transactions.
- File System: Could be used for storing transaction logs or other data locally on the ATM

UML Model for the proposed System



Procedure: -

1. **Identify the Class:** Determine the class you want to model, representing a real-world object or concept.
2. **Draw the Class Box:** Create a rectangle divided into three sections:
 - **Top Section:** Name of the class (e.g., `Person`).
 - **Middle Section:** List attributes with visibility (e.g., `- name: String`) and data types.
 - **Bottom Section:** List methods with visibility, return types, and parameters (e.g., `+ getName(): String`).
3. **Define Visibility:**
 - `+` for public, `-` for private, `#` for protected.
4. **Add Relationships:** Show interactions with other classes:
 - **Association:** Solid line (can include an arrow).
 - **Aggregation:** Hollow diamond at the line end.
 - **Composition:** Filled diamond at the line end.
 - **Inheritance:** Solid line with a hollow triangle.
 - **Dependency:** Dashed line with an arrow.
5. **Specify Multiplicity:** Add number constraints to relationships (e.g., `1..*`, `0..1`).
6. **Refine and Review:** Double-check for completeness and correctness, ensuring the diagram clearly represents class relationships and structure.

Somaiya Vidyavihar University
K J Somaiya School of Engineering

Code: -

// Sample UML class diagram for a banking system, written in Umlc.
// --- Class Definitions ---

```
class Bank {  
    String bankId;  
    String name;  
    String swiftCode;  
  
    // Methods  
    authorizeTransaction() {}  
    verifyPin() {}  
}  
  
class ATM {  
    String atmId;  
    String location;  
    String status;  
    String bankId;  
  
    // Methods  
    authenticateCard() {}  
    processTransaction() {}  
    Double checkBalance() { return 0.0; }  
}  
  
class Customer {  
    Integer id;  
    String name;  
    String address;  
    Integer phoneNo;  
    Integer accNo;  
  
    // Methods  
    withdrawMoney() {}  
    OpenAccount() {}  
    CloseAccount() {}  
    RequestCard() {}  
}  
  
class Accounts {  
    String accountNumber;  
    Double balance;  
    String accountHolderId;  
  
    // Methods  
    deposit(Double amount) {}
```

Department of Electronics and Telecommunication Engineering

Somaiya Vidyavihar University
K J Somaiya School of Engineering

```
Double getBalance() { return 0.0; }
}

class Transaction {
    String transactionId;
    Double amount;
    Date timestamp;
    String type;

    // Methods
    Boolean execute() { return true; }
    logTransaction() {}
}

class CardReader {
    Boolean isCardInserted;
    String cardNumber;

    // The diagram shows "readCard(): Card", but there is no Card class defined.
    // The return type is omitted here for simplicity as per Uml syntax.
    readCard() {}
    ejectCard() {}
}

class CashDispenser {
    Integer cashAvailable;
    String currencyType;

    // Methods
    dispenseCash() {}
    restockCash() {}
}

class ReceiptPrinter {
    Boolean paperAvailable;

    // Methods
    printReceipt(String transactionDetails) {}
}

// --- Associations between Classes ---

// A Bank is associated with one or more ATMs.
association {
    1 Bank -- 1..* ATM;
}

// An ATM is associated with one or more Customers.
// Note: This relationship in a real system might be modeled differently,
// but this reflects the provided diagram.
```

Department of Electronics and Telecommunication Engineering

Somaiya Vidyavihar University
K J Somaiya School of Engineering

```
association {
    1 ATM -- 1..* Customer;
}
// One or more Customers can have one or more Accounts (e.g., joint accounts).
association {
    1..* Customer -- 1..* Accounts;
}
// An Account is linked to one or more Transactions.
association {
    1 Accounts -- 1..* Transaction;
}

// --- Directed Associations (Dependencies) ---

// The CardReader initiates or is linked to a Transaction.
association {
    * CardReader -> * Transaction;
}
// A Transaction uses a CashDispenser to dispense cash.
association {
    * Transaction -> 1 CashDispenser;
}

// A Transaction uses a ReceiptPrinter to print a receipt.
association {
    * Transaction -> 1 ReceiptPrinter;
}
```

Somaiya Vidyavihar University
K J Somaiya School of Engineering

Observations: -

Umlple Online Draw on the right, write (Umlple) model code on the left. Analyse models and generate code.
This tool stores your data in cookies and on a server. [I understand](#). [Click to learn about privacy](#).
[Download](#) [Donate](#) For help: [User manual](#) [Ask questions](#) [Report issue](#)

Line=129 **E S S T D A M** **Generate Java** **Save & Collaborate**

Untitled x +

```
11 // Methods
12 authorizeTransaction() {}
13 verifyPin() {}
14 }
15
16 class ATM {
17     String atmId;
18     String location;
19     String status;
20     String bankId;
21
22     // Methods
23     authenticateCard() {}
24     processTransaction() {}
25     Double checkBalance() { return 0.0; }
26 }
27
28 class Customer {
29     Integer id;
30     String name;
31     String address;
32     Integer phoneNo;
33     Integer accNo;
34
35     // Methods
36     withdrawMoney() {}
37     OpenAccount() {}
38     CloseAccount() {}
39 }
```

SAVE & LOAD

TOOLS

Banking System A

DRAW

- Class
- Association
- Generalization
- Delete
- Undo
- Reindent Code
- Copy Diagram

GENERATE

Java Code

Generate It

Execute It

OPTIONS

TASKS

UML Diagram:

```
classDiagram
    class ATM {
        atmId : String
        location : String
        status : String
        bankId : String
    }
    class Bank {
        bankId : String
        name : String
        swiftCode : String
    }
    class CashDispenser {
        cashAvailable : Integer
        currencyType : String
    }
    class CardReader {
        isCardInserted : Boolean
        cardNumber : String
    }
    class Transaction {
        transactionId : String
        amount : Double
        timestamp : Date
        type : String
    }
    class ReceiptPrinter {
        paperAvailable : Boolean
    }
    class Customer {
        id : Integer
        name : String
        address : String
        phoneNo : Integer
        accNo : Integer
    }
    class Accounts {
        accountNumber : String
        balance : Double
        accountHolderId : String
    }
    ATM "1" -- "1..*" Bank
    ATM "1" -- "1..*" CashDispenser
    ATM "1" -- "1..*" CardReader
    ATM "1" -- "1..*" Transaction
    ATM "1" -- "1..*" ReceiptPrinter
    Customer "1" -- "1..*" Accounts
    Transaction "1" -- "1..*" Accounts
```

Warning on line 49: Method 'getBalance' has a name conflict with automatic generated getter/setter methods for attribute 'balance' in class 'Accounts'. Please use a different method name. [More information](#)

Conclusion:

In this experiment, we explored co-design and co-simulation for embedded system modelling using UML and FSM simulators. The process helped us understand how to design and test both hardware and software components together for optimal performance. By leveraging UML for system modelling and FSM for simulating state transitions, we learned how to validate embedded systems early in development, improving reliability and efficiency before physical implementation.

Signature of faculty in-charge with date