# Software Architecture Case Design

Lecture Notes

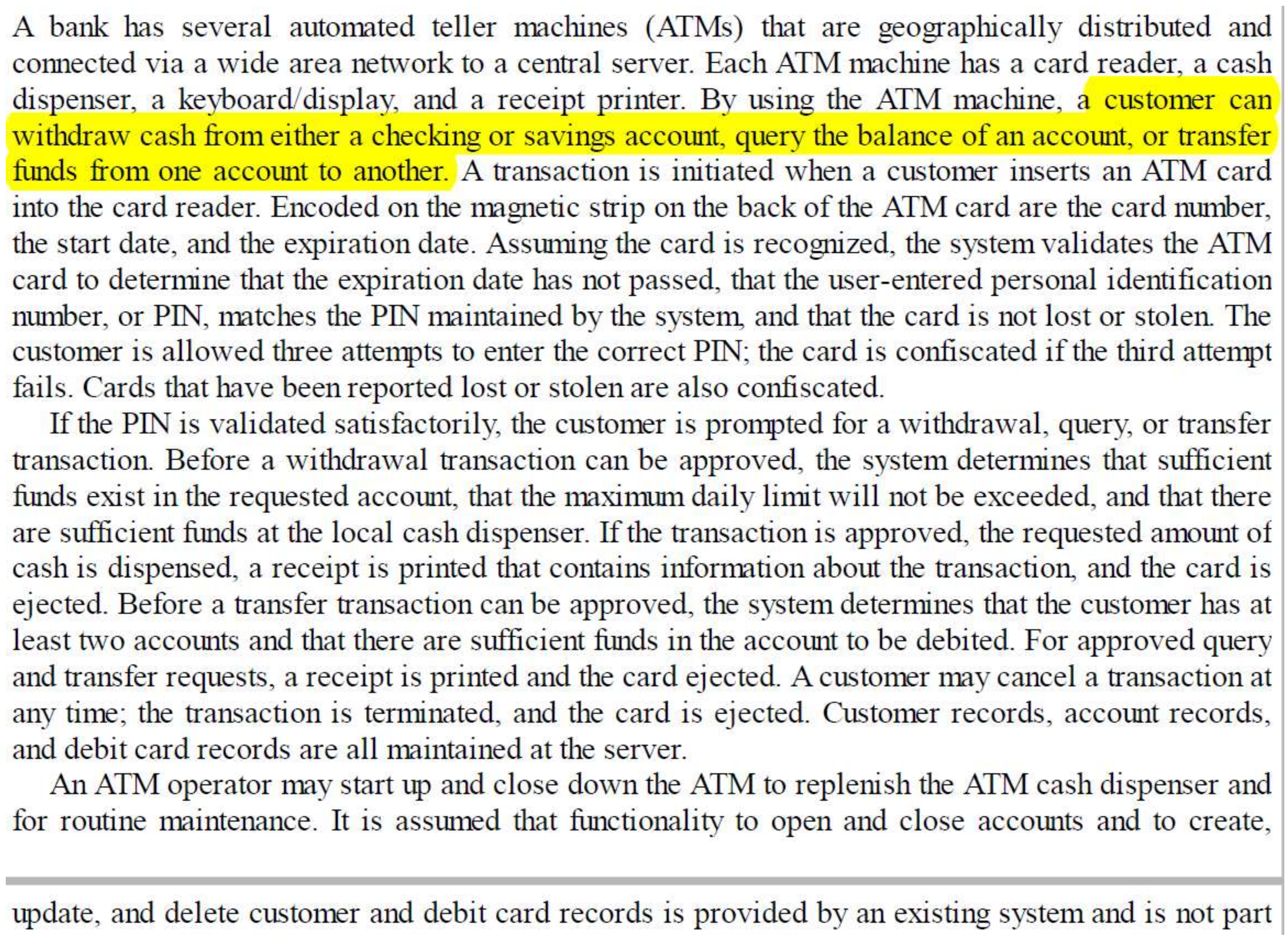
Software Design (SE324)

Prepared by Dr. Khaldoon Al-Zoubi

## Suggested Steps for Software Architecture Design

- Step1: Define Use cases & Actors
- Step2: Static Models
- Step3: Object and Class Structuring
- Step4: Dynamic Interaction Modeling

## 1: Define Use cases & Actors
## Problem Statement



A bank has several automated teller machines (ATMs) that are geographically distributed and connected via a wide area network to a central server. Each ATM machine has a card reader, a cash dispenser, a keyboard/display, and a receipt printer. By using the ATM machine, a customer can withdraw cash from either a checking or savings account, query the balance of an account, or transfer funds from one account to another. A transaction is initiated when a customer inserts an ATM card into the card reader. Encoded on the magnetic strip on the back of the ATM card are the card number, the start date, and the expiration date. Assuming the card is recognized, the system validates the ATM card to determine that the expiration date has not passed, that the user-entered personal identification number, or PIN, matches the PIN maintained by the system, and that the card is not lost or stolen. The customer is allowed three attempts to enter the correct PIN; the card is confiscated if the third attempt fails. Cards that have been reported lost or stolen are also confiscated.

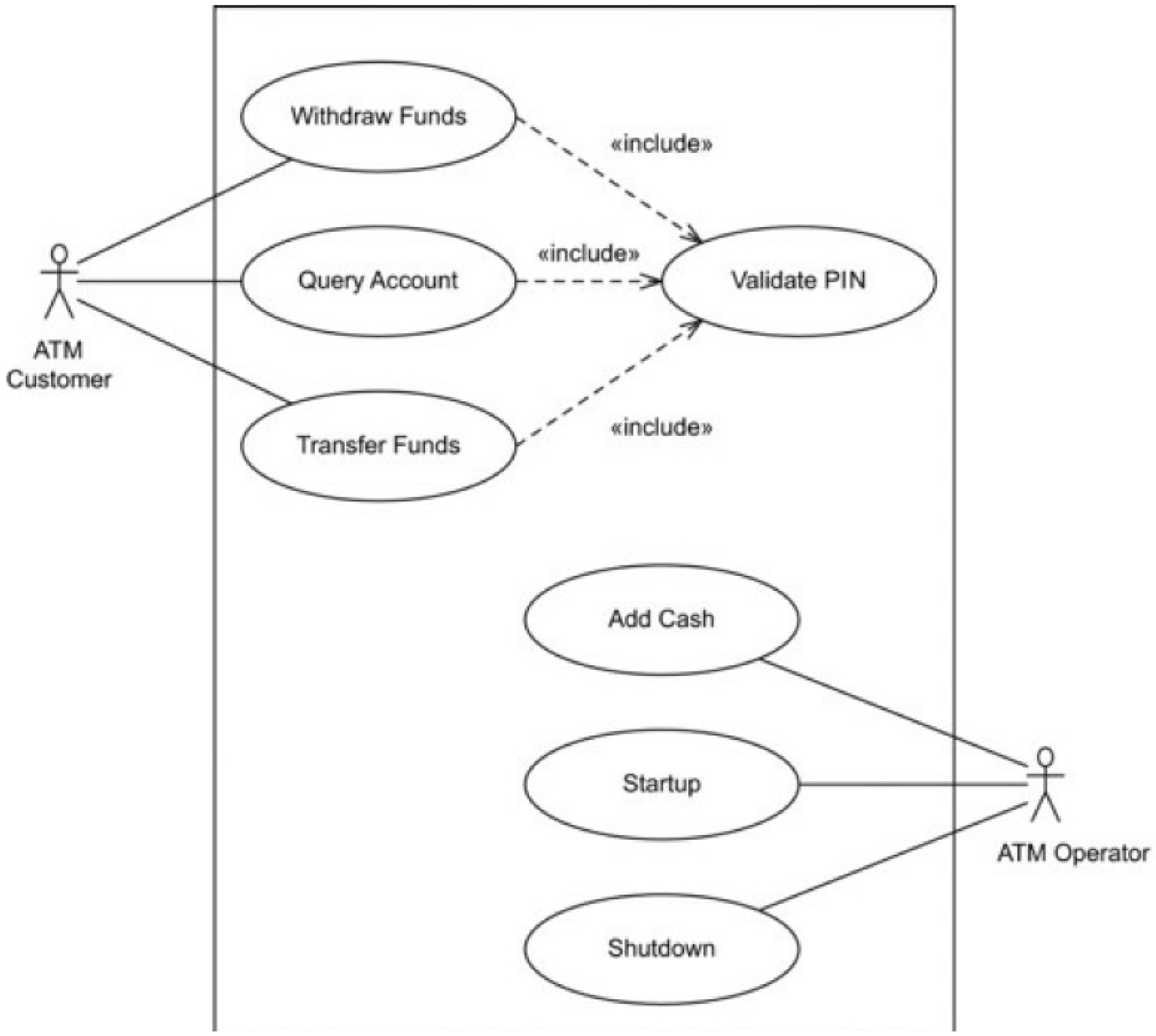
If the PIN is validated satisfactorily, the customer is prompted for a withdrawal, query, or transfer transaction. Before a withdrawal transaction can be approved, the system determines that sufficient funds exist in the requested account, that the maximum daily limit will not be exceeded, and that there are sufficient funds at the local cash dispenser. If the transaction is approved, the requested amount of cash is dispensed, a receipt is printed that contains information about the transaction, and the card is ejected. Before a transfer transaction can be approved, the system determines that the customer has at least two accounts and that there are sufficient funds in the account to be debited. For approved query and transfer requests, a receipt is printed and the card ejected. A customer may cancel a transaction at any time; the transaction is terminated, and the card is ejected. Customer records, account records, and debit card records are all maintained at the server.

An ATM operator may start up and close down the ATM to replenish the ATM cash dispenser and for routine maintenance. It is assumed that functionality to open and close accounts and to create,

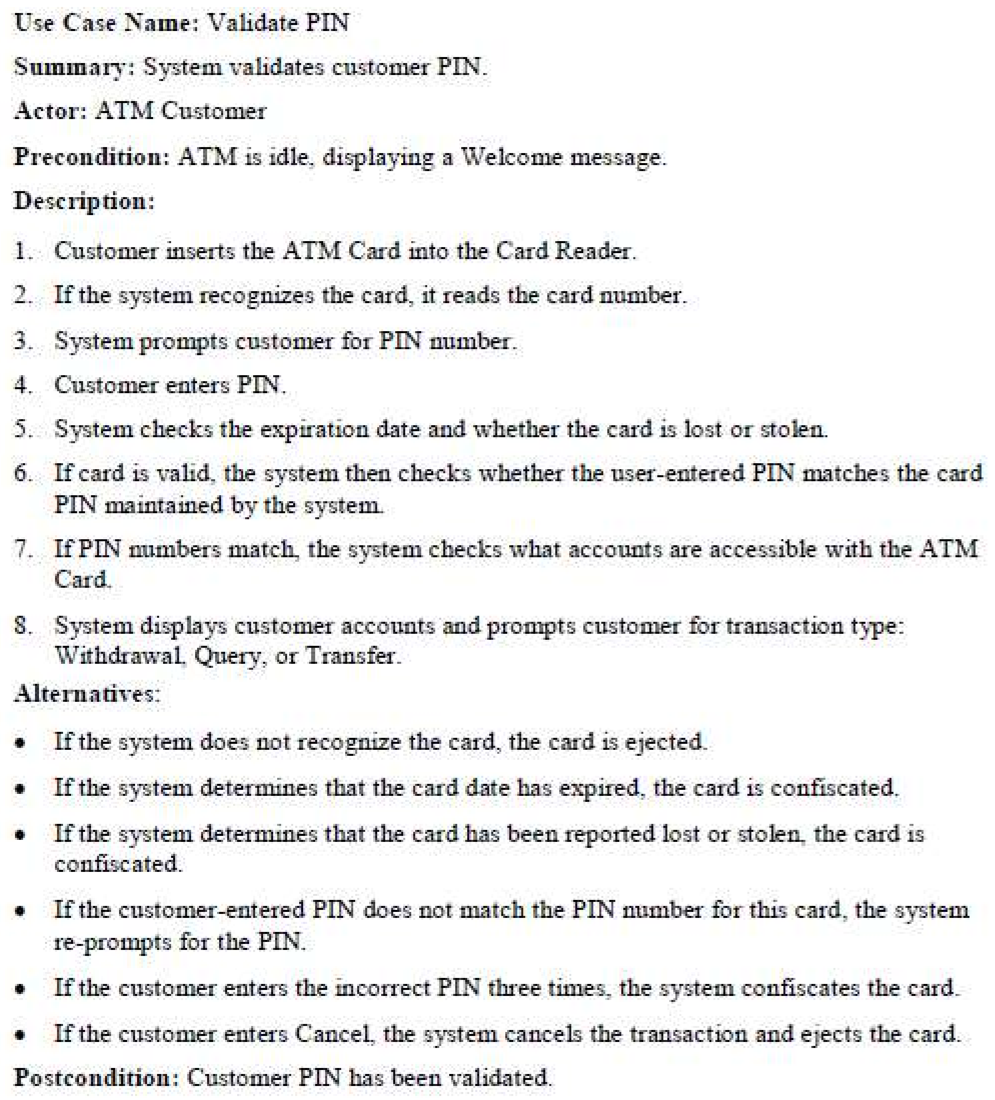
update, and delete customer and debit card records is provided by an existing system and is not part

## Requirements: Identifying Use Cases

- start by considering the actors
- Define software functional requirements in terms of use cases and actors
- Can I break Use cases into more use cases?



## Requirements: Use Case Description



**Use Case Name:** Validate PIN

**Summary:** System validates customer PIN.

**Actor:** ATM Customer

**Precondition:** ATM is idle, displaying a Welcome message.

**Description:**

1. Customer inserts the ATM Card into the Card Reader.
2. If the system recognizes the card, it reads the card number.
3. System prompts customer for PIN number.
4. Customer enters PIN.
5. System checks the expiration date and whether the card is lost or stolen.
6. If card is valid, the system then checks whether the user-entered PIN matches the card PIN maintained by the system.
7. If PIN numbers match, the system checks what accounts are accessible with the ATM Card.
8. System displays customer accounts and prompts customer for transaction type: Withdrawal, Query, or Transfer.

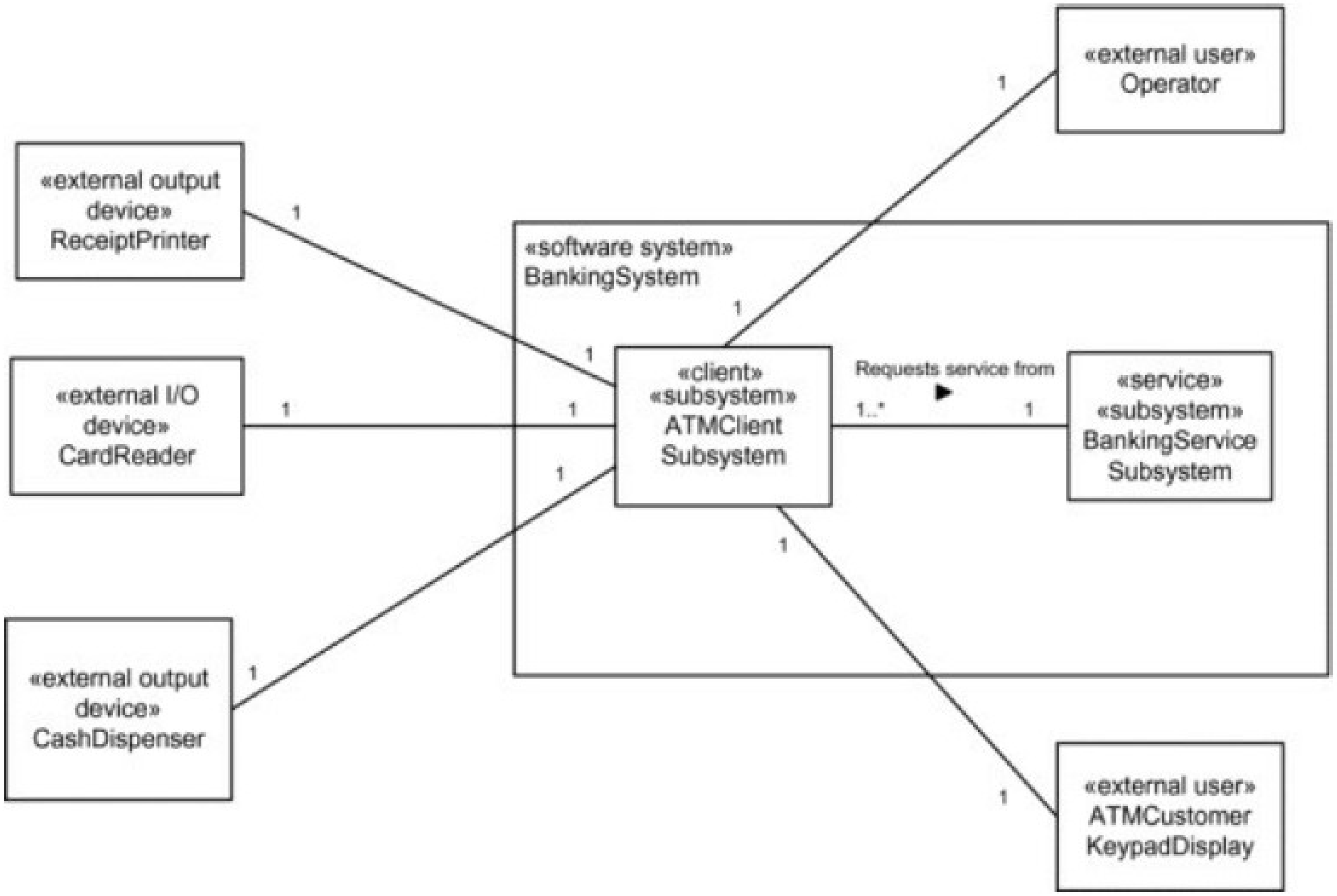
**Alternatives:**

- If the system does not recognize the card, the card is ejected.
- If the system determines that the card date has expired, the card is confiscated.
- If the system determines that the card has been reported lost or stolen, the card is confiscated.
- If the customer-entered PIN does not match the PIN number for this card, the system re-prompts for the PIN.
- If the customer enters the incorrect PIN three times, the system confiscates the card.
- If the customer enters Cancel, the system cancels the transaction and ejects the card.

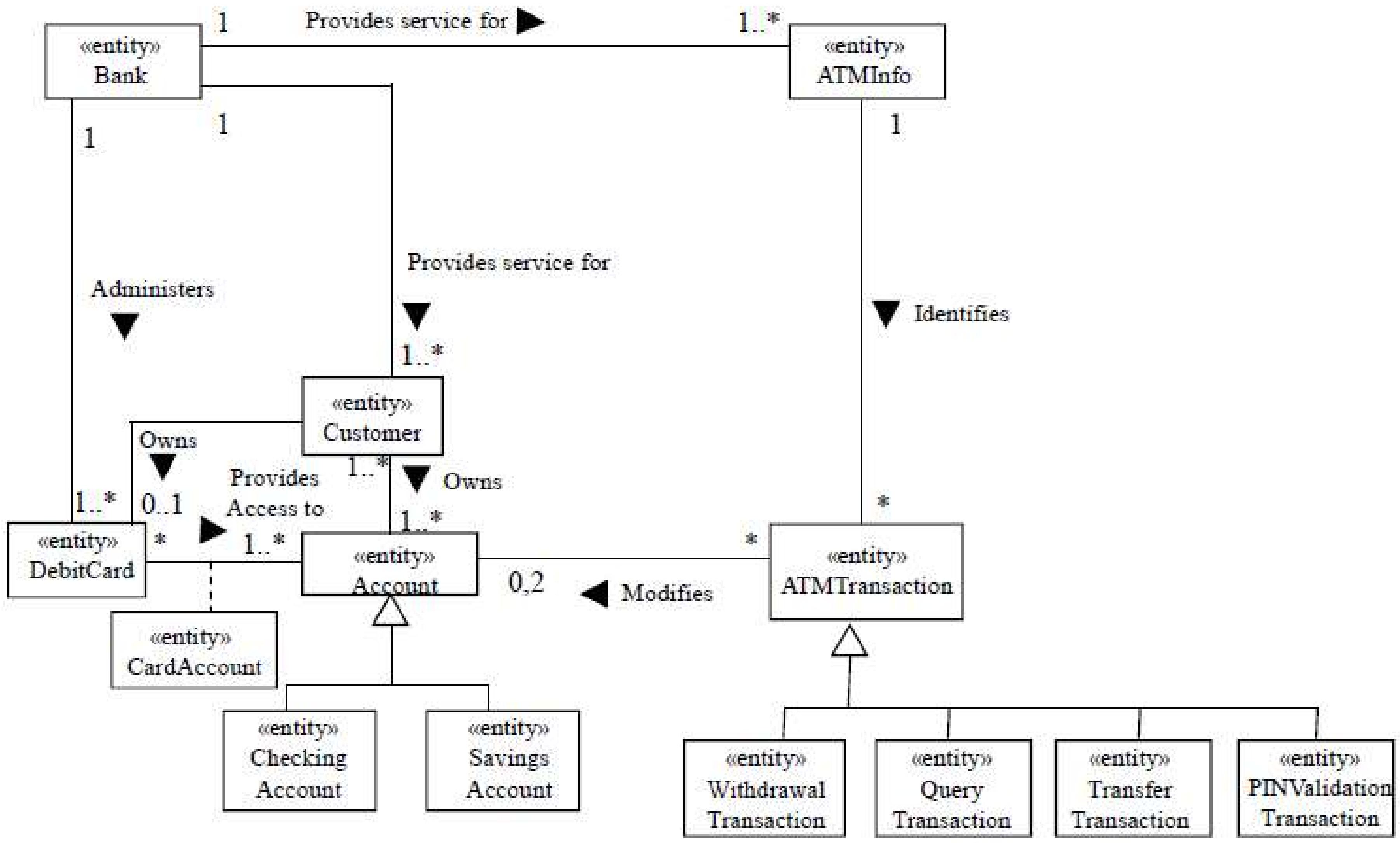
**Postcondition:** Customer PIN has been validated.

# 2: Static Modelling

## Static Modeling of the System Context

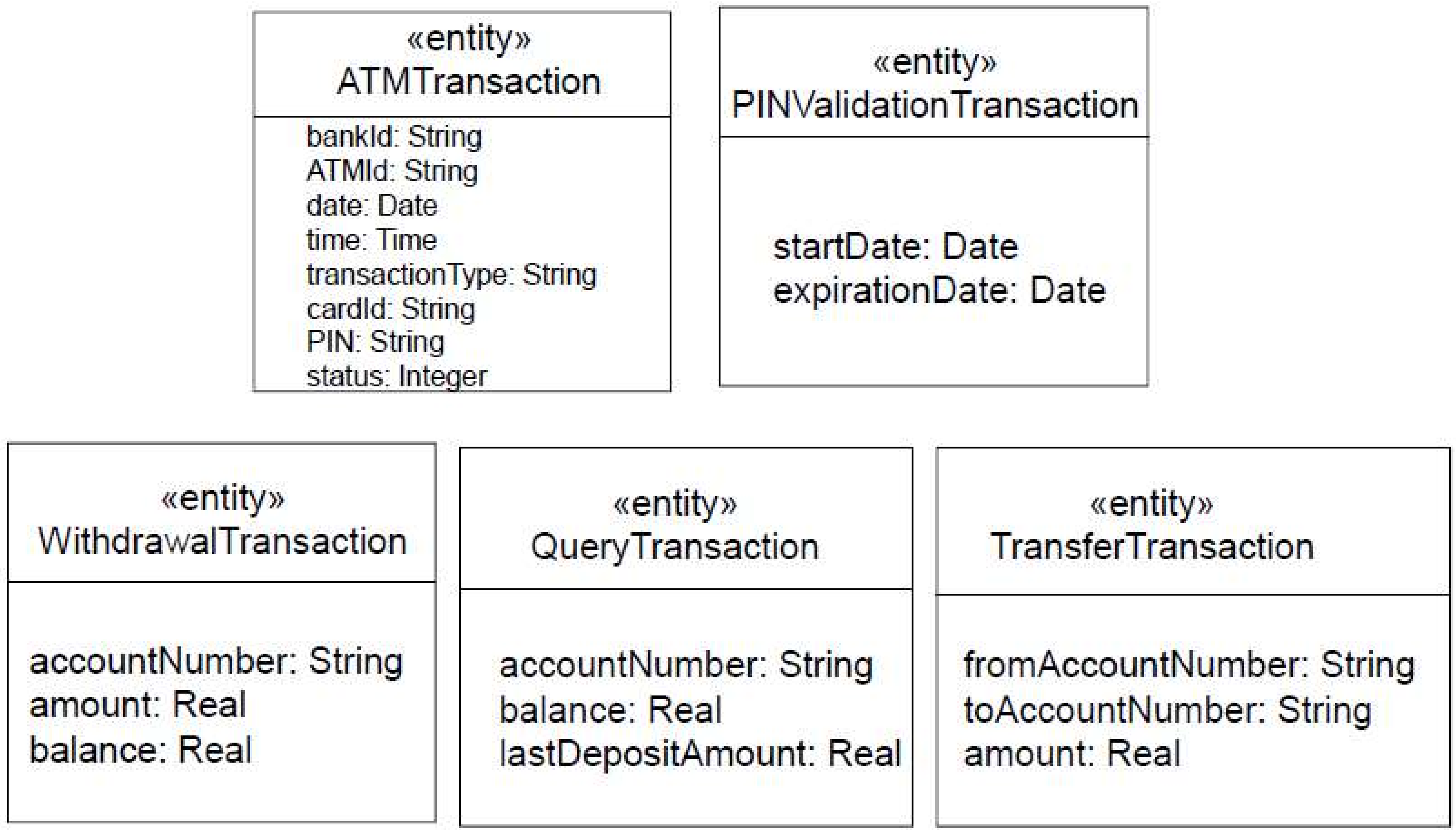


# Static Modeling for the Entity Classes (Abstracted Data Entities)

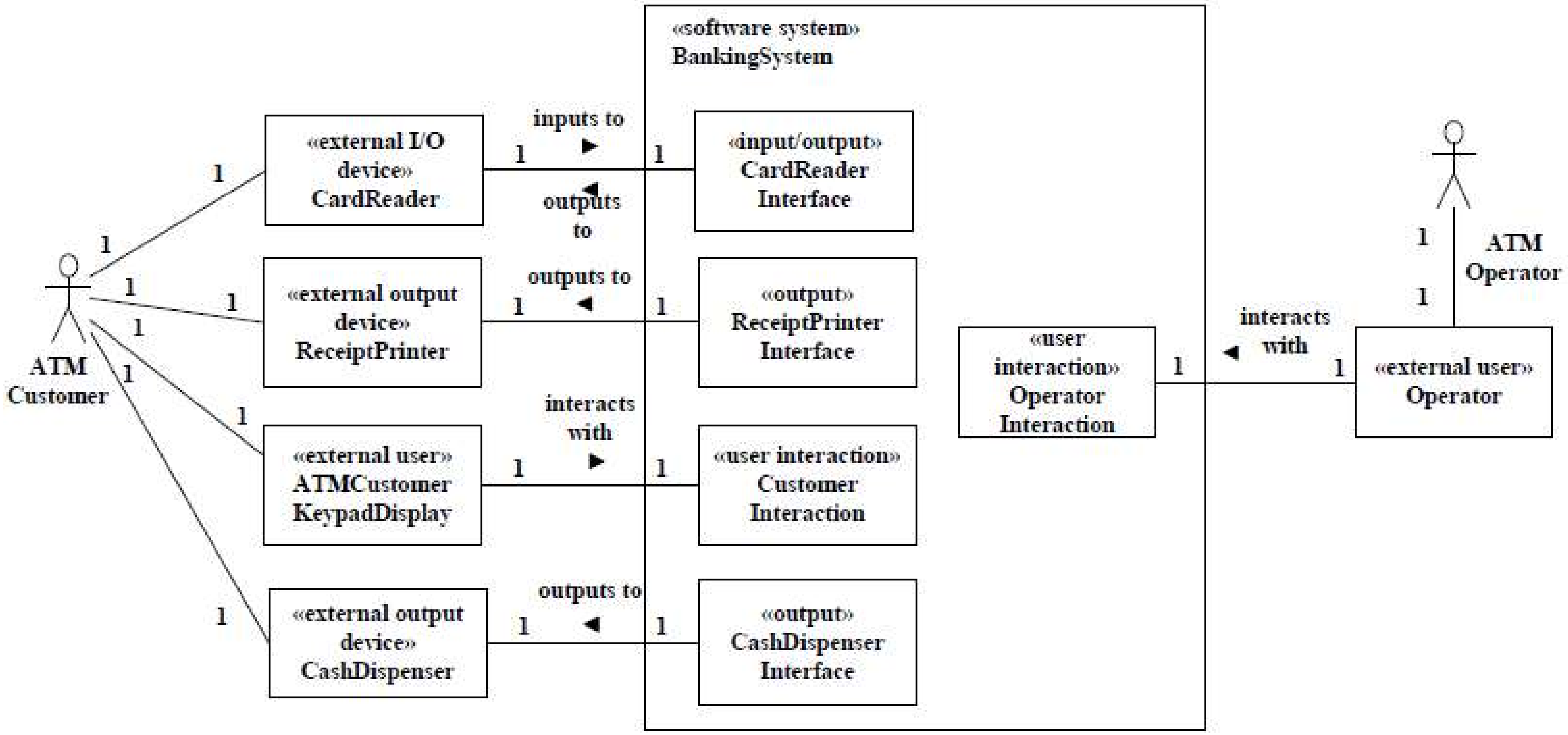


## Static Modeling of the Entity Classes (Stored Data)

- First step of encapsulation → information hiding

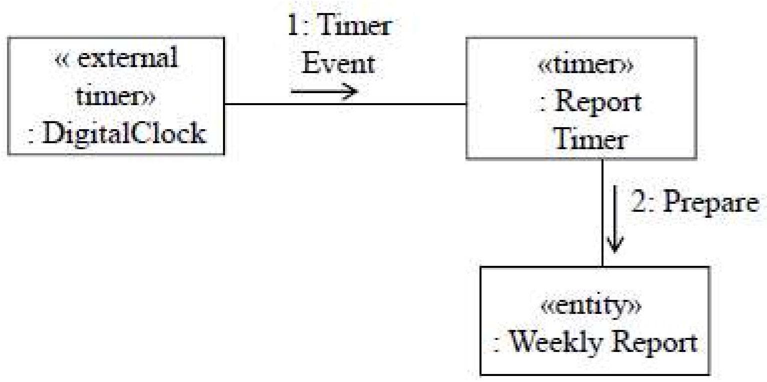


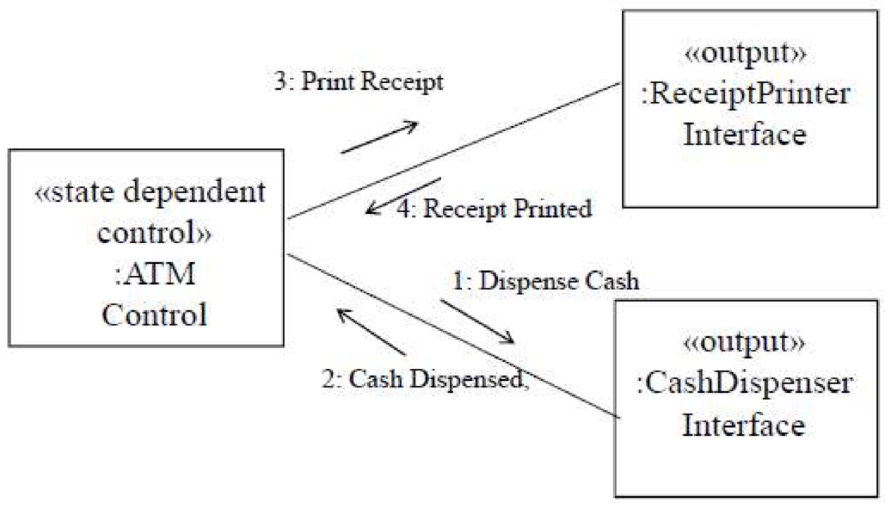
# Boundary Classes

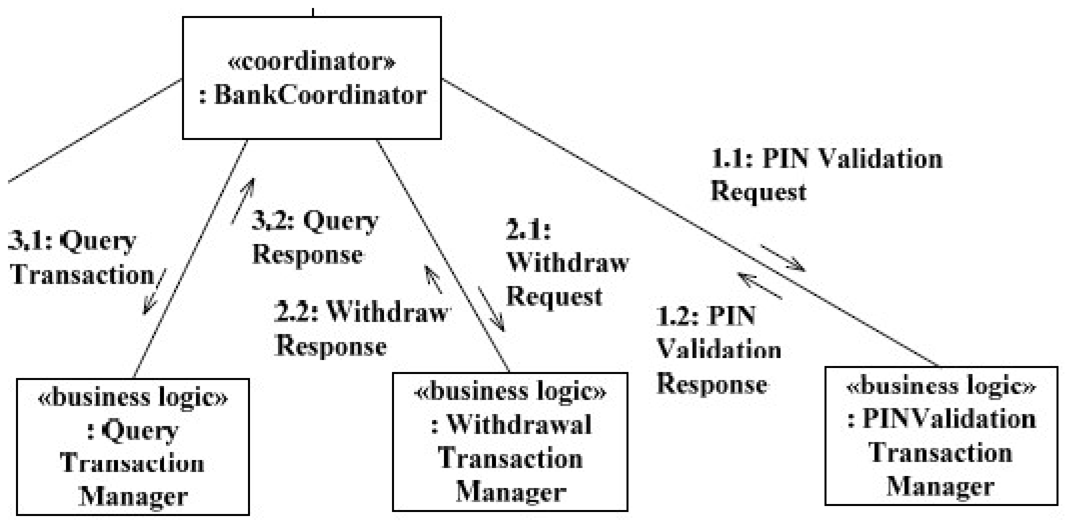

«software system» BankingSystem
«external I/O device» CardReader
«input/output» CardReader Interface
inputs to ▶
◀ outputs to
«external output device» ReceiptPrinter
«output» ReceiptPrinter Interface
outputs to ◀
«external user» ATMCustomer KeypadDisplay
«user interaction» Customer Interaction
interacts with ▶
«external output device» CashDispenser
«output» CashDispenser Interface
outputs to ◀
«user interaction» Operator Interaction
◀ interacts with
«external user» Operator
ATM Customer
ATM Operator
1

# 3: Object and Class Structuring

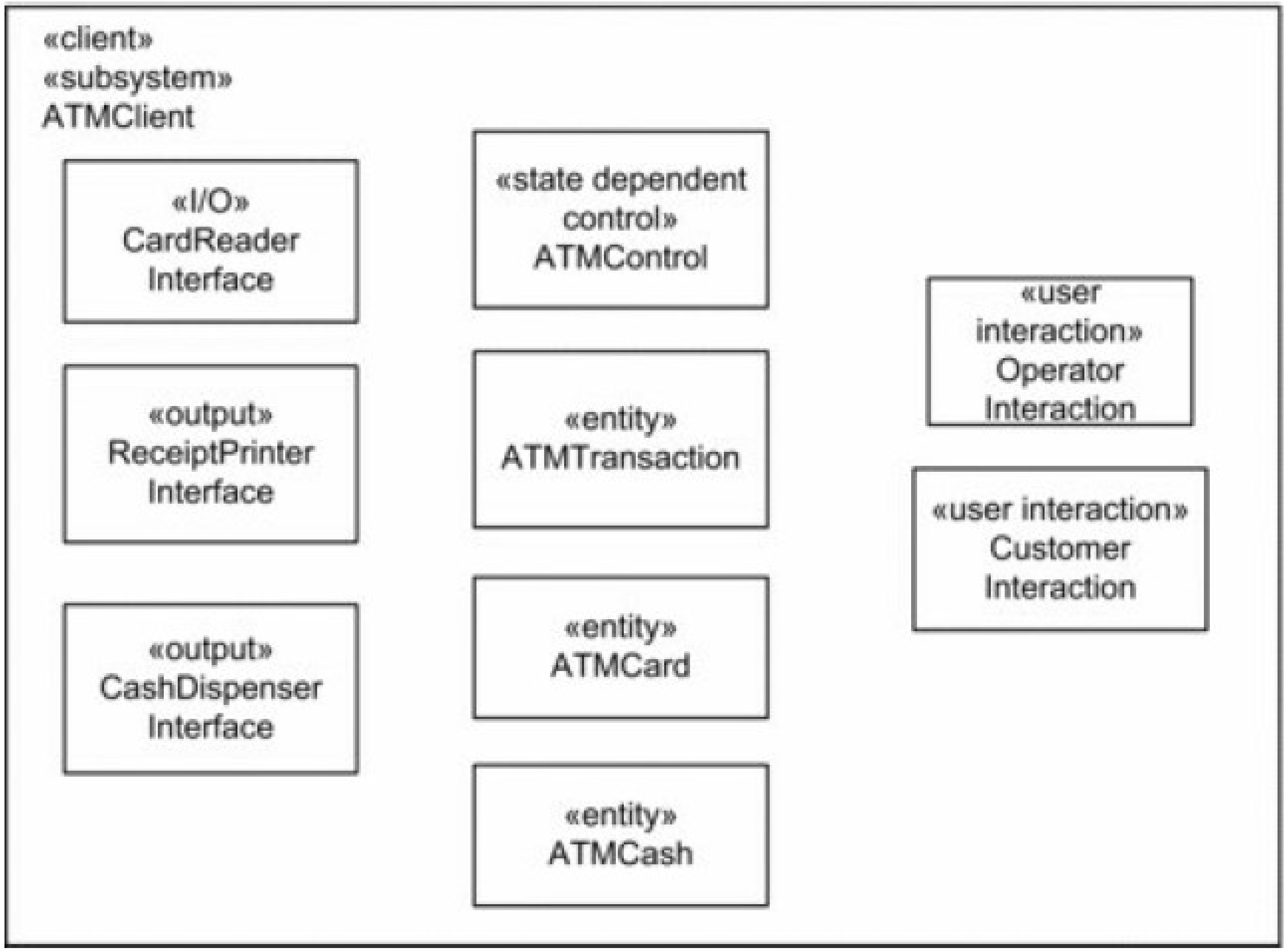
## Control objects




«state dependent control» :ATM Control
3: Print Receipt
4: Receipt Printed
«output» :ReceiptPrinter Interface
1: Dispense Cash
2: Cash Dispensed
«output» :CashDispenser Interface


«coordinator» : BankCoordinator
3.1: Query Transaction
3.2: Query Response
2.2: Withdraw Response
2.1: Withdraw Request
1.1: PIN Validation Request
1.2: PIN Validation Response
«business logic» : Query Transaction Manager
«business logic» : Withdrawal Transaction Manager
«business logic» : PINValidation Transaction Manager

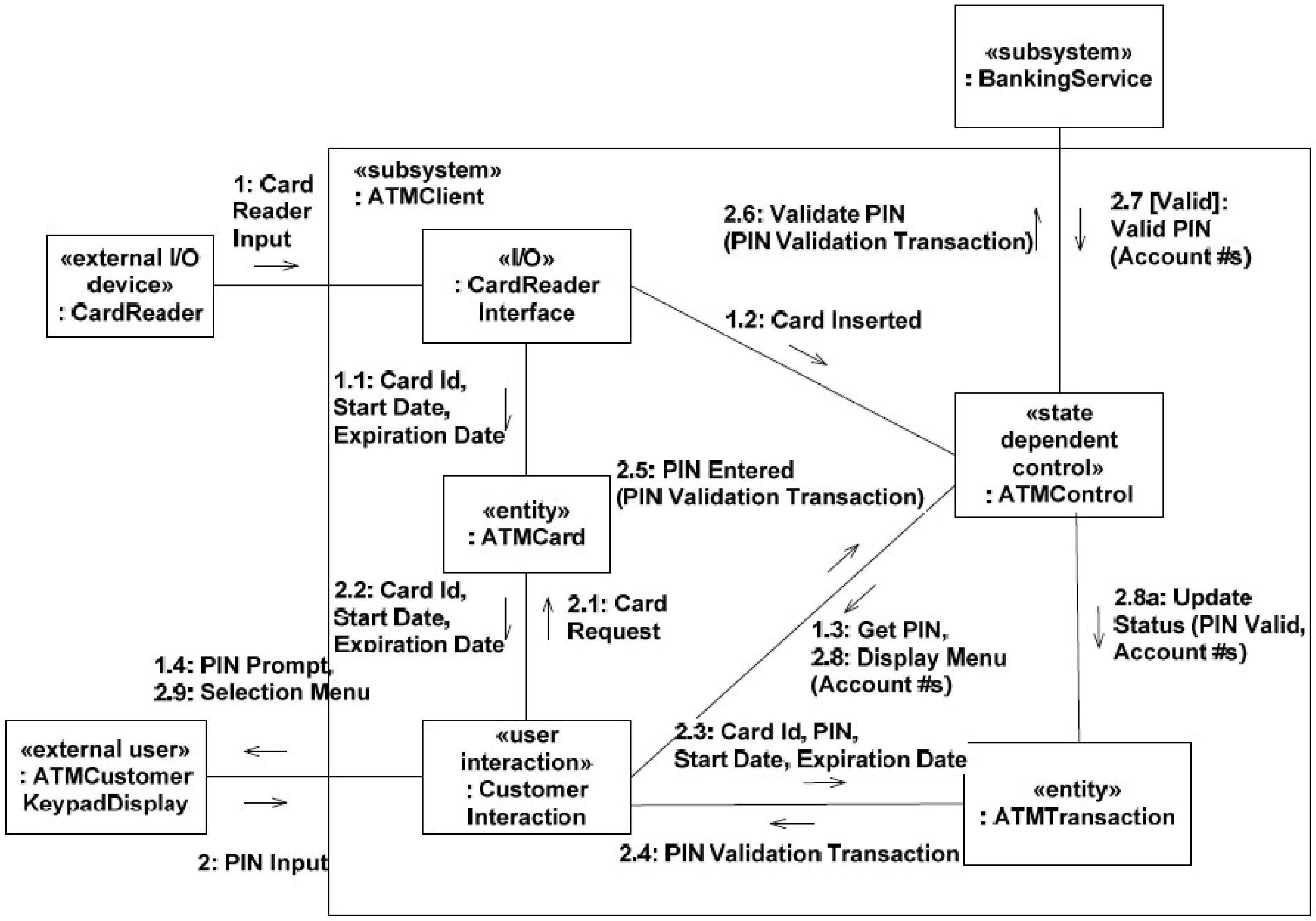
# Objects Participating in Use Cases (Client Subsystem Objects)


«client»
«subsystem»
ATMClient
«I/O» CardReader Interface
«output» ReceiptPrinter Interface
«output» CashDispenser Interface
«state dependent control» ATMControl
«entity» ATMTransaction
«entity» ATMCard
«entity» ATMCash
«user interaction» Operator Interaction
«user interaction» Customer Interaction

## 4: Dynamic Interaction Modeling

- Now: an interaction diagram is developed for each use case
  - communication diagram or a sequence diagram
- Use Cases and Scenarios
  - A scenario is one specific path through a use case
    - A particular message sequence on interaction Diagram
      - Main sequence
      - Alternative sequence

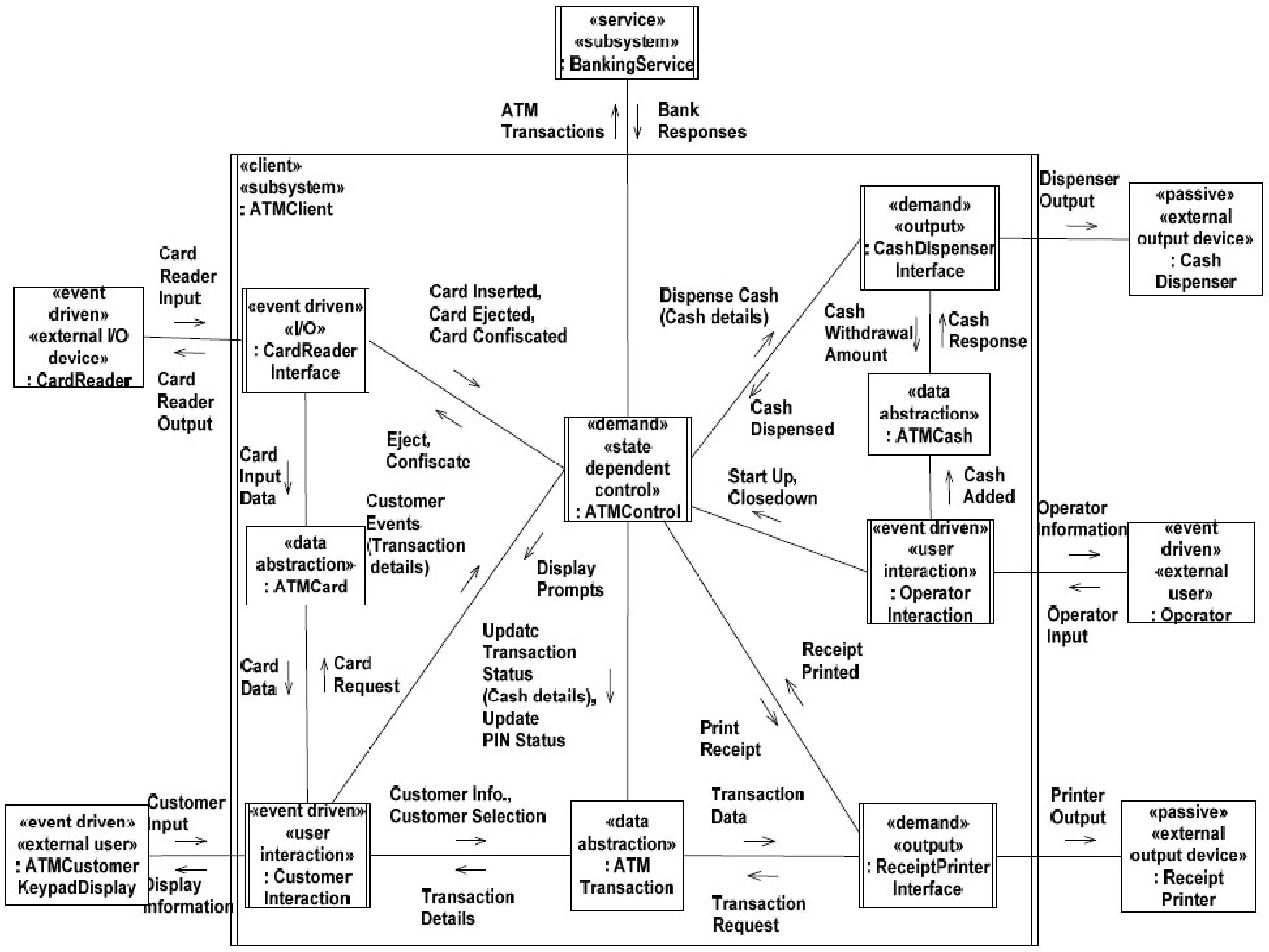
# Communication diagram: ATM client Validate PIN use case (Client subsystem)


«subsystem» : BankingService
«subsystem» : ATMClient
1: Card Reader Input
«external I/O device» : CardReader
«I/O» : CardReader Interface
2.6: Validate PIN (PIN Validation Transaction)
2.7 [Valid]: Valid PIN (Account #s)
1.2: Card Inserted
1.1: Card Id, Start Date, Expiration Date
«state dependent control» : ATMControl
2.5: PIN Entered (PIN Validation Transaction)
«entity» : ATMCard
2.2: Card Id, Start Date, Expiration Date
2.1: Card Request
1.3: Get PIN, 2.8: Display Menu (Account #s)
2.8a: Update Status (PIN Valid, Account #s)
1.4: PIN Prompt, 2.9: Selection Menu
«external user» : ATMCustomer KeypadDisplay
«user interaction» : Customer Interaction
2.3: Card Id, PIN, Start Date, Expiration Date
«entity» : ATMTransaction
2.4: PIN Validation Transaction
2: PIN Input

PIN Validation Transaction = {transactionId, transactionType, cardId, PIN, startDate, expirationDate}

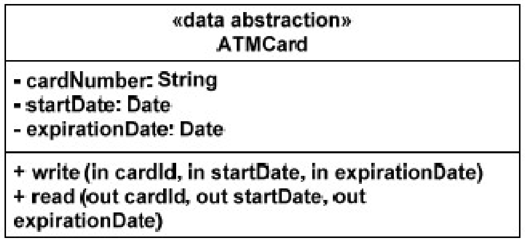
# initial consolidated concurrent communication diagram for ATM Client

- If a caller sends a **synchronous message**, it must wait until the message is done, such as invoking a subroutine. If a caller sends an **asynchronous message**, it can continue processing and doesn't have to wait for a response

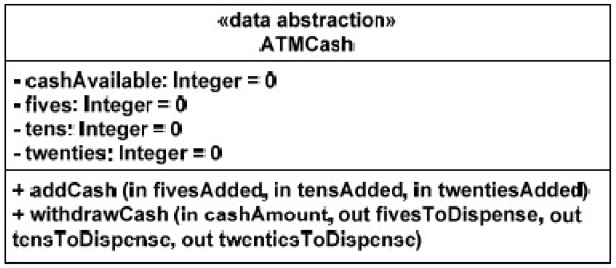

«service» «subsystem» : BankingService
ATM Transactions
Bank Responses
«client» «subsystem» : ATMClient
«event driven» «external I/O device» : CardReader
Card Reader Input
Card Reader Output
«event driven» «I/O» : CardReader Interface
Card Inserted, Card Ejected, Card Confiscated
Eject, Confiscate
Card Input Data
«data abstraction» : ATMCard
Card Data
Card Request
Customer Events (Transaction details)
«demand» «state dependent control» : ATMControl
Display Prompts
Update Transaction Status (Cash details), Update PIN Status
Dispense Cash (Cash details)
Cash Dispensed
«demand» «output» : CashDispenser Interface
Cash Withdrawal Amount
Cash Response
«data abstraction» : ATMCash
Cash Added
Dispenser Output
«passive» «external output device» : Cash Dispenser
Start Up, Closedown
«event driven» «user interaction» : Operator Interaction
Operator Information
Operator Input
«event driven» «external user» : Operator
Receipt Printed
Print Receipt
«event driven» «external user» : ATMCustomer KeypadDisplay
Customer Input
Display Information
«event driven» «user interaction» : Customer Interaction
Customer Info., Customer Selection
Transaction Details
«data abstraction» : ATM Transaction
Transaction Data
Transaction Request
«demand» «output» : ReceiptPrinter Interface
Printer Output
«passive» «external output device» : Receipt Printer

## Design Methods

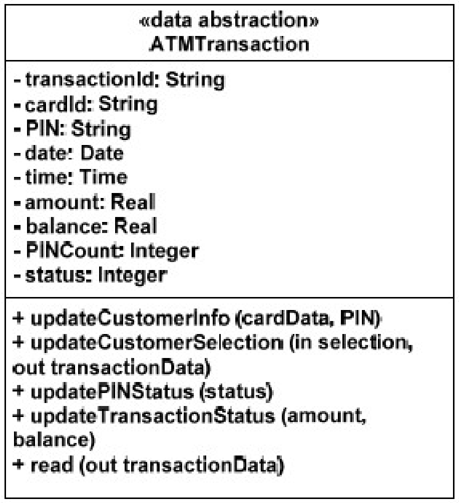
- You also need to add your interfaces & abstract classes!!!!



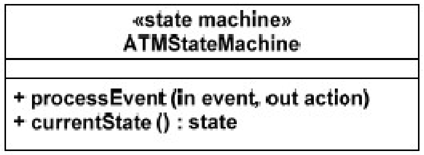
```
«data abstraction»
ATMCard
- cardNumber: String
- startDate: Date
- expirationDate: Date
+ write (in cardId, in startDate, in expirationDate)
+ read (out cardId, out startDate, out
expirationDate)
```



```
«data abstraction»
ATMCash
- cashAvailable: Integer = 0
- fives: Integer = 0
- tens: Integer = 0
- twenties: Integer = 0
+ addCash (in fivesAdded, in tensAdded, in twentiesAdded)
+ withdrawCash (in cashAmount, out fivesToDispense, out
tensToDispense, out twentiesToDispense)
```



```
«data abstraction»
ATMTransaction
- transactionId: String
- cardId: String
- PIN: String
- date: Date
- time: Time
- amount: Real
- balance: Real
- PINCount: Integer
- status: Integer
+ updateCustomerInfo (cardData, PIN)
+ updateCustomerSelection (in selection,
out transactionData)
+ updatePINStatus (status)
+ updateTransactionStatus (amount,
balance)
+ read (out transactionData)
```



```
«state machine»
ATMStateMachine
+ processEvent (in event, out action)
+ currentState () : state
```